XGBoost as a Surrogate Model for Testing Deep Reinforcement Learning Agents

Aadesh Ramai[1]
Supervisors: Dr. A. Panichella[1]**, A. Bartlett**[1]
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Aadesh Ramai
Final project course: CSE3000 Research Project
Thesis committee: Dr. A. Panichella, A. Bartlett, Dr. P. Kellnhofer

XGBoost as a Surrogate Model for Testing Deep Reinforcement Learning Agents

Abstract

Testing Deep Reinforcement Learning (DRL) agents is computationally expensive and inefficient, especially when trying to identify environment configurations where the agent fails to reach its objective. Recent work proposes the use of a Multi-Layer-Perceptron (MLP) as a surrogate model to predict whether a given environment configuration is likely to be a failing environment without running the tests. However, this raises the question of whether different surrogate models can perform this task better and whether we can improve the training of these models by fine-tuning their hyperparameters. In this work, we seek to understand how XGBoost performs as an alternative to the MLP for predicting DRL failures, together with grid search as a hyperparameter optimization technique. We evaluated our approach on the *Parking* environment from the *HighwayEnv Simulator* using a DRL agent trained using Hindsight Experience Replay (HER), where a failing environment is one in which the vehicle collides with a parked vehicle or a timeout occurs. This evaluation is based on how well the model can classify failing environments and how effective it can guide a Genetic Algorithm (GA) to find failing environments (failure search). We compared the performance of XGBoost with the MLP and found that XGBoost significantly outperforms the MLP baseline across key classification metrics such as F1-score and AUC-ROC. Furthermore, during failure search, the XGBoost-guided GA yields more failing environments with greater coverage and entropy, indicating increased diversity and effectiveness. These findings suggest that XGBoost is a strong candidate for surrogate modelling in DRL testing and offers a more reliable alternative to an MLP-based approach.

1 Introduction

Deep Reinforcement Learning (DRL) has emerged as a powerful tool for training agents that are capable of learning complex tasks by trial-and-error interaction with an environment. These environments are simulation scenarios that define the initial conditions and parameters of a test case on which the DRL agent is run. Each environment represents a single test case: the agent is run once in a specific configuration, and the outcome is observed. DRL agents typically use deep neural networks to approximate policies or value functions and have obtained exceptional results in a wide range of applications [28].

However, effective testing methods for these DRL agents remain underdeveloped. DRL agents are often trained on randomised configurations [37], meaning environments where the initial conditions are randomly sampled from a distribution, to improve generalisation and avoid overfitting to specific scenarios.

However, this approach has two major issues. Firstly, random generation is unlikely to expose failures, i.e. environments where the agent does not reach its objective, possibly resulting in an overestimation of the agent's capabilities. Secondly, finding even a challenging environment is computationally expensive: each test requires us to fully execute a simulation of the agent. In the case of *HighwayEnv* [21] for instance, the environment used in this study, a single execution of a test typically takes about 40 seconds, reaching up to 5 minutes in some cases [29]. Thorough testing requires hundreds or thousands of such test runs, substantially increasing the cost.

Recent work by Biagiola et al. [8] presents a promising alternative: using surrogate models to approximate whether the DRL agent will fail in a certain environment, with-

out actually running any tests. Their approach involves training a Multi-Layer Perceptron (MLP) on the agent's training data to predict whether a given environment configuration is likely to be a failing environment. The output of the model is then used as a fitness function in a Genetic Algorithm (GA), in order to guide it towards finding failing environment configurations. This method significantly reduces the cost of testing and provides a scalable way to explore large configuration spaces to find failing environment configurations. Although the results are promising, their study focuses exclusively on neural networks and performs minimal hyperparameter tuning, leaving open the question whether other machine learning models may perform better and whether hyperparameter optimisation techniques such as grid search [23] could improve the performance of these models.

In this paper, we seek to understand how XGBoost [9] serves as a surrogate model compared to the MLP. XGBoost works exceptionally well on tabular and structured data and is often recommended for classification when dealing with such data [33]. Furthermore, XGBoost has demonstrated strong predictive performance in modelling student learning outcomes [16] and consistently achieves top performance in machine learning competitions [5]. This makes it a strong candidate for the surrogate modelling task.

We aim to fill two specific gaps in prior work: (1) the lack of exploration beyond neural networks, and (2) the absence of extensive hyperparameter tuning in surrogate model evaluation.

Thus, we investigate the following main research question:

> **RQ1:** *How effective is XGBoost as a surrogate model to identify failing environments for a DRL agent compared to a baseline Multi-Layer-Perceptron?*

We divide this research question into two sub-questions as follows:

- **RQ1.1**: *What is the performance of XGBoost compared to the MLP in classifying failing environments?*
- **RQ1.2**: *How effective is XGBoost compared to the MLP in guiding a Genetic Algorithm to find failing environments?*

To answer these questions, we develop an initial XGBoost model and perform grid search to tune the hyperparameters. Afterwards, we obtain a final evaluation of the classification performance of both XGBoost and the MLP baseline used by Biagiola et al. [8] and compare them using five evaluation criteria: Accuracy, precision, recall, F1-Score and AUC-ROC [30]. Subsequently, we use the best-performing XGBoost configuration to guide a Genetic Algorithm (GA) to find environment configurations which the DRL agent is likely to fail on. We then run the DRL agent on these environments and run a k-means clustering algorithm [24] on the environments where the DRL agent failed to reach its objective. Afterwards, we compare their performance using three evaluation criteria: Number of failing environments produced, coverage, and entropy.

Our approach is evaluated on the *Parking* environment from the *HighwayEnv Simulator* [21] using a DRL agent trained using Hindsight Experience Replay (HER) [2], where a failing environment is one in which the vehicle collides with a parked vehicle or a timeout occurs, which means that it took too long for the agent to successfully park the car.

Our findings show that XGBoost significantly outperforms the baseline MLP in precision (0.441 vs. 0.099) F1-score (0.475 vs. 0.161), and AUC-ROC (0.820 vs. 0.687).

During failure search, the XGBoost-guided GA discovered more failing environments (17.66 vs 14.98), with greater coverage of the configuration space (82.5% vs 43.5%) and higher entropy (67.7% vs 32.1%) when comparing it with the MLP-guided GA, indicating that more diverse and meaningful environment configurations were produced.

These findings suggest that XGBoost is a strong candidate for surrogate modelling in DRL testing and offers a more reliable alternative to an MLP-based approach.

Our contributions can be summarized as follows.

- An evaluation of the performance of XGBoost in classifying failing environment configurations.
- An evaluation of the effectiveness of using XGBoost to guide a GA to find failing environment configurations.
- A comparison of the overall performance of XGBoost and MLP as a surrogate model for testing DRL agents.

2 Background and Related Work

In this section, we describe the background information necessary to fully understand this work and dive into relevant related work.

2.1 (Deep) Reinforcement Learning

Reinforcement Learning (RL) is a trial-and-error learning paradigm in which agents interact with an environment to maximize cumulative reward signals over time, typically modelled as a Markov Decision Process (MDP) [32], [35]. In this framework, the agent selects actions based on its current state to maximize expected future rewards.

Deep Reinforcement Learning (DRL) integrates deep neural networks into RL, allowing agents to learn from high-dimensional inputs such as images or sensor data [22], [11]. Algorithms such as Deep Q-Networks and policy gradients have demonstrated success in complex domains, including games and robotics [11].

However, testing DRL agents is computationally intensive. Because the agent's behaviour emerges through stochastic interaction, identifying rare failure cases requires extensive simulation across diverse environments, which often fails to yield meaningful insights [22], [11], [37].

2.2 Surrogate Modelling for Failure Prediction

Given the inefficiency of traditional DRL testing, where failures are found by running the agent through many randomized environments [37], surrogate modelling has been proposed as a solution to approximate agent behaviour [8]. In this context, a surrogate model is a classifier, trained on data collected during the agent's training, that estimates whether a DRL agent will fail in a given environment configuration without executing the actual simulation.

This approach allows for a more efficient exploration of the environment configuration space, allowing for targeted generation of environment configurations on which the DRL agent is likely to fail through guided search.

2.3 Models Used

2.3.1 Multi-Layer-Perceptron

A Multi-Layer Perceptron (MLP) is a type of feed-forward neural network consisting of an input layer, one or more hidden layers, and an output layer, where each layer is fully connected to the next. Non-linear activation functions such as ReLU or sigmoid are used to allow the model to learn complex mappings between inputs and outputs [14].

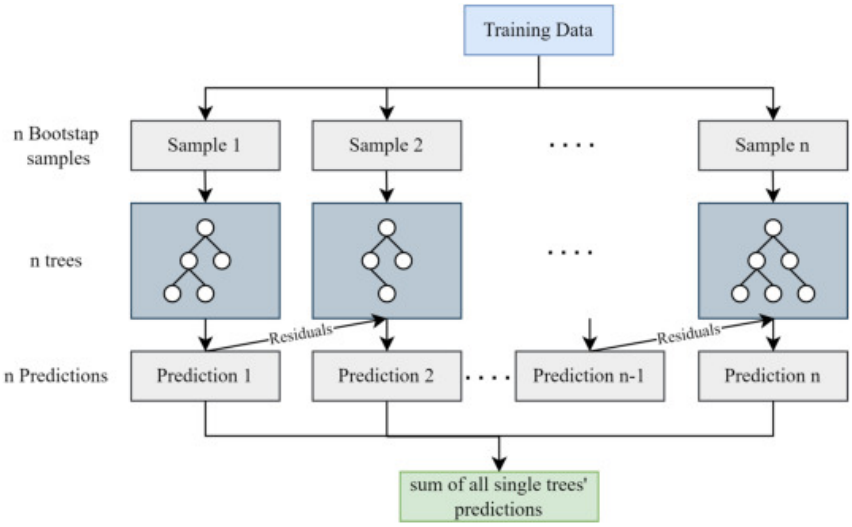

Figure 1: Overview of the XGBoost training process: At each iteration, a new tree is trained on the residual errors of the combined predictions from previous trees. [27].

MLPs have been widely applied to a variety of supervised learning tasks and are particularly effective on unstructured data such as images, text, or audio [17]. They are also relatively simple to implement and commonly used as baseline models in neural network research [14].

However, MLPs tend to underperform on structured tabular data compared to tree-based methods. This is mainly due to their sensitivity to feature scaling, initialization, and hyperparameter settings, as well as their limited ability to model feature interactions without substantial data or architectural tuning [33], [6]. This reduces their suitability in domains such as DRL testing, where the input consists of structured environment configurations, and the dataset is often imbalanced or small.

2.3.2 XGBoost

XGBoost (Extreme Gradient Boosting) is a machine learning algorithm based on the principle of gradient boosting, which constructs an ensemble of decision trees in a sequential manner. Figure 1 shows an overview of the XGBoost training process, where each boosting step, a new tree is trained to predict the residual errors of the current model. The model uses bootstrap samples from the training data and aggregates the predictions of all individual trees to form the final output, typically by summation. What sets XGBoost apart from traditional boosting methods is its use of first- and second-order derivatives during optimization, allowing more accurate and efficient updates to the model during training [9].

XGBoost incorporates several techniques that enhance performance and generalization:

- **Regularization**: L1 (lasso) and L2 (ridge) regularization terms are included in the objective function to penalize the complexity of the model, which helps to prevent overfitting.
- **Tree Pruning**: Instead of growing trees greedily, XGBoost uses a depth-first approach and prunes branches that do not significantly improve the loss.

- **Handling Missing Data**: XGBoost automatically learns the best direction to take when encountering missing values in the data, improving its robustness in real-world scenarios.

These features make XGBoost particularly powerful on structured, tabular datasets [33], where it consistently achieves top performance in machine learning competitions [5]. Therefore, we have chosen to use XGBoost in this study.

2.4 Genetic Algorithm in Failure Search

A Genetic Algorithm (GA) is a population-based optimization technique inspired by the principles of natural selection. It generates an initial population of candidate solutions and iteratively applies selection, crossover, and mutation to evolve better solutions over time [39], [26]. In each generation, solutions are evaluated using a fitness function and the most promising ones are used to produce the next generation.

In the context of failure search for Deep Reinforcement Learning (DRL), the above-mentioned solutions are environment configurations on which the DRL agent is likely to fail. GA's are guided towards these solutions using a surrogate model as the fitness function, which predicts whether a given configuration will lead to failure without having to run a full simulation [8].

Saliency further improves this process. Saliency methods identify which input features most influence the model's predictions. In this case, the saliency maps generated by the surrogate model highlight which environment parameters have the strongest effect on the predicted probability of failure [3], [4]. By biasing mutations and crossover toward these salient parameters, the GA can focus on parts of the search space where failures are more likely, resulting in a more efficient and diverse exploration compared to purely random search.

2.5 Related Work

Work by Bhatt et al. [7] proposed DSAGE, a framework that uses a deep surrogate model to estimate the behaviour of the DRL agent in procedurally generated environments. The surrogate model is used to guide the generation of diverse test cases without running the agent. While their method focuses on behavioural diversity, it does not aim to identify failure cases.

Altmann et al.'s work [1] proposed a method to select representative DRL trajectories using surrogate-based metrics that prioritize behavioural diversity and certainty. Their goal is to support interpretability by identifying trajectories that are useful to explain agent behaviour. However, they do not generate new environments or evaluate the likelihood of failure.

Sieusahai et al. [34] trained surrogate models to replicate the behavior of DRL agents in Atari environments, with the aim of improving interpretability. Their models were designed to explain the agent's decisions by imitating its policy based on previously collected data. However, they do not use the surrogate to guide testing or generate new environment configurations.

The work by Biagiola et al. [8] is the most directly relevant to this study. They proposed a search-based testing approach for DRL agents, implemented in a tool called Indago. Their method involves training a Multi-Layer-Perceptron (MLP) on interaction data from the training process of DRL agents. This classifier serves as a surrogate model that predicts the likelihood of agent failure in unseen environment configurations, which is then used to guide a search algorithm toward possibly failing environment configurations. Their approach

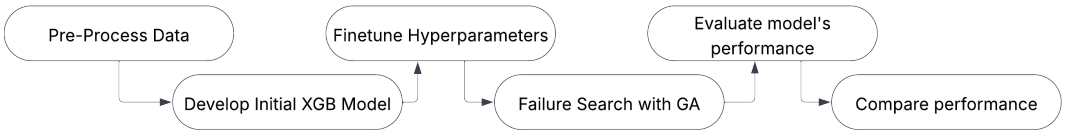

Figure 2: Flow of the proposed methodology.

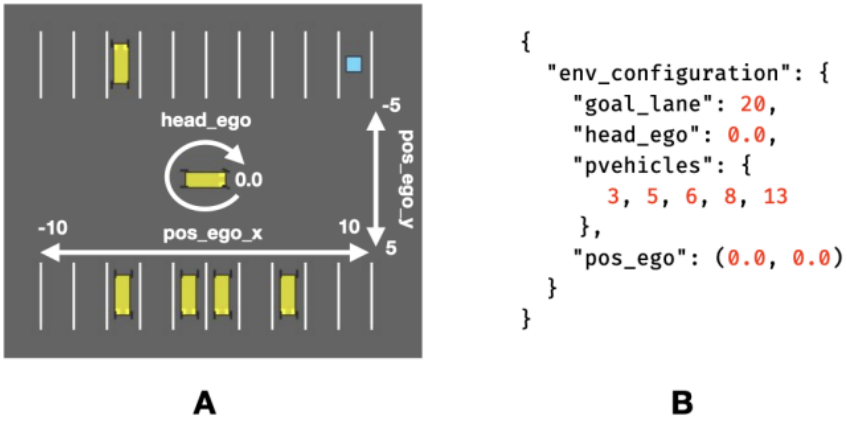

Figure 3: Example environment configuration in the Parking environment. (A) Visual layout of the environment, showing the ego vehicle, goal lane, and parked vehicles. (B) Corresponding configuration file encoding initial conditions for the surrogate model [8].

demonstrated that the search-based method could uncover significantly more failures, and more diverse ones, compared to random testing or baseline sampling approaches [37]. However, they focused solely on their MLP and performed minimal hyperparameter tuning. Our work builds on these findings by investigating whether XGBoost can outperform MLPs in this context, particularly when combined with a hyperparameter tuning technique like grid search [23].

3 Proposed Methodology

In this work, we propose a methodology for testing Deep Reinforcement Learning (DRL) agents that builds on the surrogate modelling approach introduced by Biagiola et al. [8]. Figure 2 presents the flow of this methodology.

3.1 Data Collection and Preprocessing

The foundation of our surrogate modelling approach is a dataset originally generated by Biagiola et al. [8]. They derived this from the interaction history of a DRL agent trained using Hindsight Experience Replay (HER) [2] with environments from the *Parking* environment in the *HighwayEnv* simulator [21].

Each datapoint consists of an environment configuration on which the DRL agent was trained (see Figure 3), and a binary label representing a failing or successful environment configuration. In order to make this data easier to work with, it is pre-processed to a simple array of numbers, where one-hot-encoding is used to represent free and taken parking spots.

The environment seen in Figure 3 would look as following: [*0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 (pvehicles one-hot-encoded), 20 (goal_lane), 0.0 (head_ego), 0.0, 0.0 (pos_ego)*]

In the initial phases of training, the DRL agent performs poorly in nearly all environment configurations due to the lack of a learned policy. These early failures are not particularly informative for distinguishing failing from successful environment configurations and could bias the surrogate model. We therefore omit the first x % of the training samples, where x is a hyperparameter that will be tuned.

Since failures are much rarer than successes in later training stages, the dataset is imbalanced. To mitigate the impact of this imbalance, we applied and compared two different techniques. The first is weighted sampling, where the probability of a failing environment being sampled during training is much higher than a successful one. The second is oversampling, where the failing environments are duplicated such that the ratio of failing/successful environments is equal to a specified input, which is a hyperparameter that will be tuned.

Finally, we split up the dataset into a training, validation, and test set.

3.2 XGBoost as Surrogate Model

The input features for XGBoost consist of the aforementioned array representation of an environment configuration, chosen since it fully characterizes a single starting environment. We use the training set to train our XGBoost model, using the validation set to calculate the validation loss at each boosting step. This allows us to implement early stopping, i.e. stop the training of the model when the validation loss does not improve for a certain amount of iterations. The loss function used is *logloss*.

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right]$$

Where p_i is the predicted probability of the positive class and $y_i \in \{0, 1\}$ is the class label.

We use XGBoost as our surrogate model, based on how well it has been proven to perform on tabular structured data [33], similar to our input.

Furthermore, XGBoost's support for regularization and instance weighting allows us to easily manage overfitting and to account for the rarity of failure cases during training, without the need to perform extensive data augmentation techniques [9]. Furthermore, XGBoost has demonstrated strong predictive performance in modelling student learning outcomes [16] and consistently achieves top performance in machine learning competitions [5], making it a strong candidate for the surrogate modelling task.

3.3 Grid Search

We extend the surrogate modelling pipeline with hyperparameter optimization: we use grid search [23] to fine-tune XGBoost's hyperparameters. The hyperparameters considered can be found in table 1.

Due to time constraints, it was not feasible to perform an exhaustive grid search over all possible hyperparameter configurations. Instead, we opted for a round-based approach: the hyperparameters were divided into several groups, indicated by a number, and grid search was conducted within each group while keeping the remaining hyperparameters fixed. After each round, we kept the best performing configuration, based on its F1-score, from that group and repeated the process for the next group. This procedure continued until all hyperparameters were tuned. The performance of each configuration was averaged over 5

Hyperparameter	Description	Values	Group
Test split	Ratio of data split into training and validation sets	[0.1, 0.2, 0.3, 0.4, 0.5]	1
Learning rate	Shrinking rate of feature weights at each boosting step	[0.1, 0.2, 0.3, 0.4, 0.5]	1
Max Depth	Maximum depth of each tree	[1, 2, 5, 10, 20]	1
Max Leaves	Maximum number of leaves per tree	[0, 16, 32, 64, 128]	1
Gamma	Minimum loss reduction to partition a leaf	[0, 0.1, 0.5, 1, 5]	2
Alpha	L1 regularization term	[0, 0.1, 0.5, 1, 5]	2
Lambda	L2 regularization term	[0, 0.1, 0.2, 0.5, 1]	2
Subsample	Subsample ratio of training instances	[0.6, 0.7, 0.8, 0.9, 1]	3
Colsample By Tree	Subsample ratio of columns per tree	[0.6, 0.7, 0.8, 0.9, 1]	3
Training progress filter	Percentage of training data omitted before training	[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]	3
Patience	Trees added without improvement before early stopping	[1, 2, 5, 10, 20, 30]	3
Minimum child weight	Minimum sum of instance weight (Hessian) per child	[1, 2, 4, 16, 32, 64, 128]	4
Oversample	Ratio of failing to successful environments in oversampling	[0, 0.25, 0.5, 0.75, 1]	4
Weight Loss	Whether to apply class weights during training	[True, False]	4

Table 1: Grid search hyperparameters with descriptions, value ranges, and groupings.

different seeds to account for variance in the training and validation split. Grid search was selected because it is simple to implement, easy to reproduce, and well-suited to problems with a limited number of hyperparameters. It provides a clear and structured way to explore different combinations of parameters and was appropriate for the scale and goals of this study. Although more advanced methods such as random search [6] or Bayesian optimization [12] can be more efficient, grid search was a practical and appropriate choice given the experimental constraints.

3.4 Guided Failure Search using Genetic Algorithms (GA)

Once trained, the surrogate model is integrated into a failure search algorithm. Similarly to the approach by Biagiola et al. [8], we use the surrogate's output as a fitness function in a Genetic Algorithm (GA) to guide the generation of new environment configurations that

are likely to cause the DRL agent to fail. The GA evolves the environment parameters by selecting, mutating, and crossing over candidate configurations, with XGBoost calculating the failure likelihood for each environment. The DRL agent is then run on each environment and the failure or success of the agent is recorded.

By prioritizing configurations that are likely to fail based on surrogate predictions, we minimize expensive simulations with the actual DRL agent. This not only improves efficiency, but also increases the likelihood of discovering rare or diverse failures.

4 Study Design

In this work, we investigate the following main research question:

> **RQ1:** *How effective is XGBoost as a surrogate model to identify failing environments for a DRL agent compared to a baseline Multi-Layer-Perceptron?*

We divide this research question into two sub-questions as follows:

- **RQ1.1**: *What is the performance of XGBoost compared to the MLP in classifying failing environments?*
- **RQ1.2**: *How effective is XGBoost compared to the MLP in guiding a Genetic Algorithm to find failing environments?*

To answer RQ1, we assess the classification performance of the XGBoost model in predicting whether a given environment configuration results in a failure for the DRL agent. We train the surrogate model on labelled environment data and evaluate it using standard classification metrics: accuracy, precision, recall, F1-score, and AUC-ROC [30]. The model's hyperparameters are tuned using grid search, and the results are compared with a baseline MLP-based classifier used by Biagiola et al. [8].

To answer RQ2, we integrate the trained XGBoost model into a Genetic Algorithm (GA) to guide the generation of new environment configurations. The surrogate model serves as a fitness function, scoring candidate configurations based on the likelihood that the DRL agent fails on it. Subsequently, we run the DRL agent trained using Hindsight Experience Replay (HER) [2] on each environment to see whether the agent actually fails in these environments. Finally, we apply a k-means clustering algorithm [24] to the generated environment configurations that caused the agent to fail, using the silhouette score to find the optimal number of clusters.

We then measure the model's effectiveness during this process using three metrics: the number of failing environments discovered, the coverage of the search space, and the entropy of the produced configurations. Coverage measures how well the discovered environments span the search space. It is defined as the percentage of unique clusters that contain at least one failing environment produced by the GA. The diversity quantifies how uniformly the failures are distributed across these clusters. It is calculated based on the proportion of failures that fall into each cluster and normalized to range between 0% (all environments in one cluster) and 100% (uniform distribution across clusters). These metrics help us not only evaluate the amount of failures discovered, but also how effectively we explore the search space.

Table 2: Selected hyperparameters for XGBoost based on grid search.

Hyperparameter	**Value**
Test split	0.1
Learning rate	0.4
Max depth	5
Max leaves	0
Subsample	0.85
Colsample by tree	0.9
Training progress filter	25
Gamma	0
Lambda	0.1
Patience	2
Alpha	5
Oversample	0
Min child weight	1
Weight Loss	True

4.1 Dataset

The dataset used consists of 8,790 data points of which approximately 2% are configurations where the agent failed to reach its objective. Each datapoint consists of the following:

- Ego position: $x \in [-10, 10],\ y \in [-5, 5]$
- Ego heading: a scalar in $[0.0, 1.0)$
- Goal lane: integer $\in [1, 20]$
- Parked lane occupancy: one-hot-encoded array of size 20
- Outcome: binary label indicating success or failure

The dataset was first divided into two subsets, with 80% used for training and validation, and the remaining 20% reserved for testing. This split is very common and often shows the best results [13]. Moreover, based on the small size of our data-set, any split higher than this could have negative impact on the model's performance. Within the training and validation set, 90% was used for training and 10% for validation. This split was part of our grid search.

4.2 Parameter settings

Table 2 shows the parameters used to train our XGBoost model, selected through grid search. We set the number of boosting iterations to 50, because we noticed that early stopping consistently occurred before the 50th iteration.

To obtain the classification performance, we evaluated the performance of XGBoost and a pre-trained MLP, whose parameters are described in the work by Biagiola et al. [8]. We evaluated the performance of both models five different times, with seeds 15 through 19 used to create the held-out set, and obtained the mean and standard deviation of the results.

To obtain the performance of the models during failure search, we ran the Genetic Algorithm using XGBoost and a pre-trained MLP from Biagiola et al. 50 times with a randomised seed, each run generating 50 different environment configurations.

4.3 Evaluation Criteria

For RQ1, we evaluate the performance of the model using five commonly used classification metrics: **Accuracy**, **Precision**, **Recall**, **F1-score**, and **AUC-ROC** [30].

- **Accuracy** measures the proportion of correctly classified instances among all predictions
- **Precision** measures the proportion of positive identifications that were actually correct
- **Recall** measures the proportion of actual positives that were correctly identified
- **F1-score** is the harmonic mean of precision and recall
- **AUC-ROC** measures the model's ability to distinguish between classes. A higher AUC indicates better discrimination.

For RQ2, we evaluate the performance of the model using the following metrics.

- **Amount of failing environments produced**
- **Coverage**: How well the produced environment configurations span the search space.

$$\text{Coverage} = \frac{k}{|\mathcal{C}_{\text{total}}|} \times 100$$

  Where k is the number of unique clusters that contain at least one environment configuration generated by the model being evaluated and C_{total} is the total number of clusters created.

- **Entropy**: The diversity of the produced environment configurations.

$$\text{Entropy} = -\left(\sum_{i=1}^{k} p_i \log_2(p_i)\right) \Big/ \log_2(k) \times 100$$

  Where p_i is the proportion of the model's produced environment configurations that fall into cluster i.

To measure the statistical significance of the difference between the evaluation metrics of both models, we compute the Mann-Whitney U test [25]. To measure the effect size, we compute the Vargha-Delaney metric $\hat{A}_{12}$ [38].

5 Results

In this section, we discuss the results of our research questions separately.

5.1 XGBoost classification performance

Table 3 shows the comparative performance between XGBoost and the MLP for accuracy, precision, recall, F1-score, and AUC-ROC.

XGBoost achieves a much higher precision (0.441 ± 0.063) than the MLP (0.099 ± 0.015), with a statistically significant p-value ($p = 0.0079$) and a maximum effect size ($\hat{A}_{12} = 1$). This indicates that XGBoost produces substantially fewer false positives.

Recall is slightly higher for XGBoost (0.525 ± 0.036) than for the MLP (0.424 ± 0.070), but the difference is not statistically significant ($p = 0.0749$, $\hat{A}_{12} = 0.860$). This suggests a comparable ability to detect actual failing environments, although XGBoost still performs marginally better.

Table 3: Comparison between the classification performance of XGBoost and the baseline MLP. For each evaluation metric, the mean over 5 different seeds is shown together with the standard deviation.

Metric	**XGB**	**MLP**	**p-value**	**Effect Size** ($\hat{A}_{12}$)
Accuracy	**0.944** $\pm$ 0.010	0.788 $\pm$ 0.027	0.0119	1
Precision	**0.441** $\pm$ 0.063	0.099 $\pm$ 0.015	0.0079	1
Recall	**0.525** $\pm$ 0.036	0.424 $\pm$ 0.070	0.0749	0.860
F1-score	**0.475** $\pm$ 0.034	0.161 $\pm$ 0.022	0.0079	1
AUC-ROC	**0.820** $\pm$ 0.021	0.687 $\pm$ 0.023	0.0079	1

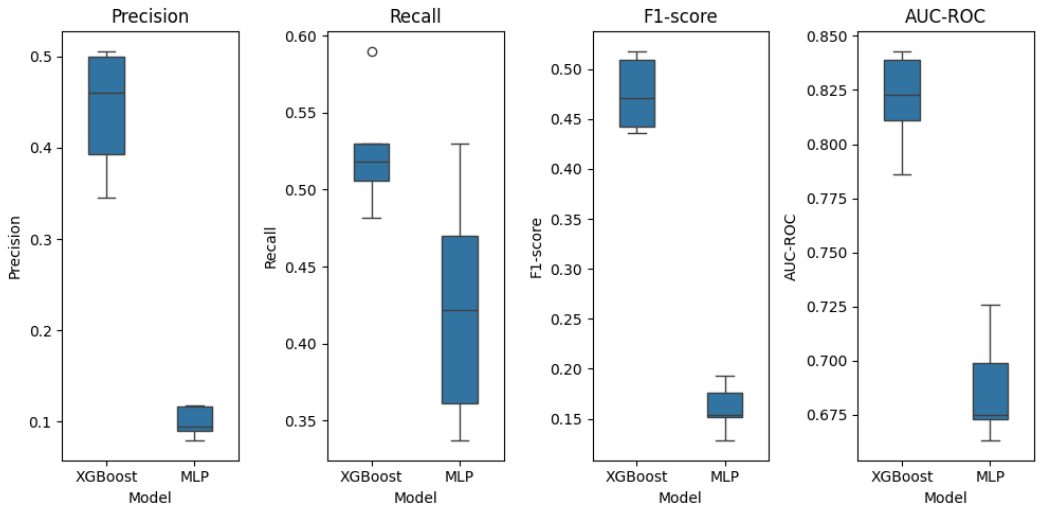

Figure 4: Boxplots comparing XGBoost and MLP accross 4 different evaluation metrics for classification

The F1-score is significantly higher for XGBoost (0.475 ± 0.034) compared to MLP (0.161 ± 0.022), with $p = 0.0079$ and $\hat{A}_{12} = 1$. This indicates a stronger and more reliable classification performance overall.

In terms of AUC-ROC, XGBoost also outperforms the MLP (0.820 ± 0.021 vs. 0.687 ± 0.023), with $p = 0.0079$ and $\hat{A}_{12} = 1$. This shows that the classifier is more effective at separating the two classes across all thresholds.

Accuracy is higher for XGBoost (0.944 ± 0.010) than for MLP (0.788 ± 0.027), with $p = 0.0119$ and $\hat{A}_{12} = 1$, although this metric is less informative due to the class imbalance.

Figure 4 displays boxplots for precision, recall, F1-score and AUC-ROC. Although XGBoost outperforms the MLP across all metrics on average, it shows greater variability between runs. This is particularly evident in the wider interquartile ranges for precision and F1-score. These results indicate that, while XGBoost provides better predictive performance, it may be more sensitive to changes in the test split.

Table 4: Mean and standard deviation of the number of failing environments produced, amount of clusters produced, the coverage of the search space and the entropy for the baseline MLP and XGBoost model across 50 runs.

Metric	**XGB**	**MLP**	**p-value**	**Effect Size ($\hat{A}_{12}$)**
Failing environments	$\mathbf{17.66} \pm 3.37$	14.98 ± 3.24	0.003	0.709
Coverage	$\mathbf{82.52} \pm 16.63$	43.49 ± 9.03	7.03×10^{-17}	0.9784
Entropy	$\mathbf{67.69} \pm 18.59$	32.12 ± 26.94	2.10×10^{-10}	0.8676

5.2 XGBoost failure search performance

Table 4 shows the comparative performance between XGBoost and MLP in guiding a Genetic Algorithm to discover failing environment configurations.

On average, XGBoost finds more failing environments (17.66 ± 3.37) than MLP (14.98 ± 3.24), with a statistically significant difference ($p = 0.003$) and a moderate effect size ($\hat{A}_{12} = 0.709$). This suggests that the surrogate model is more effective in directing the search toward likely failures.

Coverage of the configuration space is also substantially higher with XGBoost ($82.52\% \pm 16.63$) than with MLP ($43.49\% \pm 9.03$), with $p < 0.00001$ and a very large effect size ($\hat{A}_{12} = 0.9784$). This indicates that the failures discovered by XGBoost span a much wider range of clusters, resulting in better exploration.

Entropy is also higher for XGBoost ($67.69\% \pm 18.59$ vs. $32.12\% \pm 26.94$), with strong statistical significance ($p < 0.00001$) and a large effect size ($\hat{A}_{12} = 0.8676$). This suggests that the generated failures are not only more widespread, but also more diverse.

Together, these findings show that XGBoost is not only capable of identifying more failure cases, but also enables broader and more balanced coverage of the environment configuration space during failure search.

6 Threats to Validity

Several threats to validity can be identified for our study. In the next subsections, we discuss the threats and how we addressed them.

6.1 Internal Validity

Both surrogate models were trained on the same dataset and evaluated using identical performance metrics and random seeds. However, a potential threat to internal validity may arise from the difference in the extent of hyperparameter tuning. Although the pre-trained MLP did undergo some tuning [8], such as filtering level and number of hidden layers, this was relatively limited compared to the more thorough grid search applied to XGBoost. As a result, the comparison may be biased in favour of XGBoost due to the more extensive optimisation it received.

6.2 External Validity

The experiments were carried out in a single environment, using a single DRL agent and a single Genetic Algorithm. As such, the generalizability of the findings to other DRL algorithms, other search strategies, and other environments remains uncertain. Expanding the evaluation to include multiple agents [15] [20], other search strategies beyond Genetic

Algorithms [19] [10], and other environments [36] would provide a stronger basis for generalisation.

6.3 Construct Validity

A threat to construct validity may come from the definition of DRL agent failure. In this study, failure is defined as the inability of the DRL agent to reach its objective within the simulation environment, due to either a collision or a timeout. Although this is clear and measurable, it may not capture all types of meaningful failures that could occur in real-world deployments, such as unsafe behaviours that technically still achieve the objective.

6.4 Conclusion Validity

Similarly to the previously mentioned threat to internal validity, we cannot certainly conclude XGBoost is a better alternative than the baseline MLP, since the extent of the hyperparameter optimisation techniques applied to both models are not comparable. Furthermore, we do not know how their performance will compare in any environment other than the *Parking* environment in the *HighwayEnv Simulator* [21] used in this study.

7 Conclusion and Future Work

This study explored the use of XGBoost as a surrogate model for identifying failure cases in Deep Reinforcement Learning (DRL) agents. In contrast to previous work [8] using a Multi-Layer Perceptron (MLP), we investigated whether XGBoost, combined with grid search, could yield improved performance in both failure classification and guided failure search.

To address this, we evaluated XGBoost on three fronts: its performance in classifying failing environments, its ability to guide a Genetic Algorithm to discover such environments, and its effectiveness relative to the baseline MLP used by Biagiola et al. [8].

The results show that XGBoost consistently outperforms the baseline MLP across several classification metrics such as precision, F1-score, and AUC-ROC, while maintaining a similar recall. In the context of failure search, XGBoost guided the Genetic Algorithm to produce more failing environments, with significantly better coverage and entropy, indicating greater diversity and distribution in the discovered failures. Statistical tests confirm that these differences are significant with large effect sizes.

Despite these promising results, there are several directions for future work. Firstly, while this study focused exclusively on XGBoost, other tree-based models such as LightGBM [18] or CatBoost [31] could be explored for potential improvements. Furthermore, the MLP baseline was minimally subjected to hyperparameter tuning, which may have disadvantaged it when comparing it to XGBoost. Incorporating a tuned MLP, as a result of an extensive grid search, as a baseline would provide a more balanced comparison.

Future work could also expand the evaluation to multiple DRL agents, such as SAC [15] and TQC [20], and alternative search strategies beyond Genetic Algorithms, like simulated annealing [19] or Particle Swarm Optimisation (PSO) [10], helping to assess the generalizability of the findings.

In conclusion, this work demonstrates that XGBoost is a strong candidate for surrogate modelling in the context of DRL testing, offering a practical and effective alternative to neural network-based approaches.

8 Responsible Research

This section will discuss the ethical considerations that are relevant to this work and will describe its reproducibility.

8.1 Ethical Considerations

This work aims to improve the efficiency of testing Deep Reinforcement Learning (DRL) using surrogate models. Since DRL systems are increasingly being deployed in high-stakes domains such as autonomous driving and robotics, enhancing their reliability is of clear ethical importance. By helping to uncover potential failures more effectively, this research contributes positively to the responsible development of AI technologies.

However, surrogate models are approximations and may mispredict rare or adversarial failures. It is therefore important that such models are used to complement, not replace, thorough testing methods. This research does not involve human participants, sensitive data, or foreseeable environmental risks. All datasets were synthesized in simulation environments, ensuring compliance with ethical standards.

8.2 Reproducibility

Reproducibility was a key design consideration in this research. The experiments were carried out using publicly available tools and libraries. All datasets were generated from controlled simulation environments using fixed seeds and configuration parameters.

Hyperparameters, training procedures, and evaluation metrics are clearly documented. The codebase used for training and testing the surrogate models is publicly available on Github[1], allowing other researchers to reproduce and verify the results with minimal setup.

8.3 Use of AI

Large Language Models (LLMs) were used to improve the clarity and flow of the wording, assist in locating sources, support layout formatting in Overleaf, and help with debugging code.

References

[1] Philipp Altmann, Celine Davignon, Maximilian Zorn, Fabian Ritz, Claudia Linnhoff-Popien, and Thomas Gabor. Surrogate fitness metrics for interpretable reinforcement learning, 2025.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2018.

[3] Shahin Atakishiyev, Mohammad Salameh, Hengshuai Yao, and Randy Goebel. Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions, 2024.

[4] Akanksha Atrey, Kaleigh Clary, and David Jensen. Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning, 2020.

[5] Candice Bentejac, Anna Csorgo, and Gonzalo Martinez-Munoz. A comparative analysis of xgboost, 11 2019.

[1] https://github.com/TheCrazyRecker/surrogate-reloaded-xgboost

[6] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[7] Varun Bhatt, Bryon Tjanaka, Matthew C. Fontaine, and Stefanos Nikolaidis. Deep surrogate assisted generation of environments, 2022.

[8] Matteo Biagiola and Paolo Tonella. Testing of deep reinforcement learning agents with surrogate models. *ACM Transactions on Software Engineering and Methodology*, 33(3):1–33, March 2024.

[9] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 16, pages 785–794. ACM, August 2016.

[10] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[11] Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018.

[12] Peter I. Frazier. A tutorial on bayesian optimization, 2018.

[13] Abbas Gholamy, Vladik Kreinovich, and Olga Kosheleva. Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation. Technical Report UTEP-CS-18-09, 2018.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[16] Soukaina Hakkal and Ayoub Ait Lahcen. Xgboost to enhance learner performance prediction. *Computers and Education: Artificial Intelligence*, 7:100254, 2024.

[17] Simon S. Haykin. *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.

[18] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[20] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics, 2020.

[21] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[22] Yuxi Li. Deep reinforcement learning: An overview, 2018.

[23] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas, 2019.

[24] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[25] H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50 – 60, 1947.

[26] Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag Berlin Heidelberg, New York, 3rd rev. extended edition, 1996. Includes bibliographical references and index.

[27] Amirhossein Mohammadi, Shaghayegh Karimzadeh, Seyed Amir Banimahd, Volkan Ozsarac, and Paulo B. Lourenco. The potential of region-specific machine-learning-based ground motion models: Application to turkey. *Soil Dynamics and Earthquake Engineering*, 172:108008, 2023.

[28] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Deep reinforcement learning: An overview. In Yaxin Bi, Supriya Kapoor, and Rahul Bhatia, editors, *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*, pages 426–440, Cham, 2018. Springer International Publishing.

[29] Sandy Nugroho, De Rosal Ignatius Moses Setiadi, and Hussain Md Mehedul Islam. Exploring dqn-based reinforcement learning in autonomous highway navigation performance under high- traffic conditions. *Journal of Computing Theories and Applications*, 2:54–66, 02 2024.

[30] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, 2020.

[31] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[32] Martin L. Puterman. Chapter 8 markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pages 331–434. Elsevier, 1990.

[33] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need, 2021.

[34] Alexander Sieusahai and Matthew Guzdial. Explaining deep reinforcement learning agents in the atari domain through a surrogate model, 2021.

[35] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[36] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[37] Jonathan Uesato, Ananya Kumar, Csaba Szepesvari, Tom Erez, Avraham Ruderman, Keith Anderson, Krishmamurthy, Dvijotham, Nicolas Heess, and Pushmeet Kohli. Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures, 2018.

[38] Andras Vargha and Harold Delaney. A critique and improvement of the "cl" common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics - J EDUC BEHAV STAT*, 25, 06 2000.

[39] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, June 1994.