Surrogate Reloaded: Fast Testing for Deep Reinforcement Learning with Bayesian Neural Networks

Rodrigo Montero González[1]
Supervisor(s): Dr. Annibale Panichella[1], **Antony Bartlett**[1]
[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: Rodrigo Montero González
Final project course: CSE3000 Research Project
Thesis committee: Dr. Annibale Panichella, Antony Bartlett, Dr. Petr Kellnhofer

Surrogate Reloaded: Fast Testing for Deep Reinforcement Learning with Bayesian Neural Networks

Abstract

Deep Reinforcement Learning (DRL) is a powerful framework for training autonomous agents in complex environments. However, testing these agents is still prohibitively expensive due to the need for extensive simulations and the rarity of failure events, such as collisions or timeouts, where the agent fails to complete its task safely or correctly. Existing surrogate models, such as Multi-Layer Perceptrons (MLPs), are a promising improvement by predicting failures without requiring full simulation runs. However, prior research has focused almost exclusively on MLPs, leaving it unclear whether other, more expressive machine learning models could improve performance. In this paper, we investigate whether Bayesian Neural Networks (BNNs), which incorporate probabilistic reasoning into neural architectures, can be more effective surrogates for failure prediction in DRL environments. We developed, trained, and evaluated a BNN surrogate and compared it against a pre-trained MLP baseline, using the HighwayEnv car parking environment as our test case. Our evaluation focused on comparing the predictive accuracy, precision, recall, F1-score, and Area Under the Receiver Operating Characteristic Curve (AUC-ROC) using training data, as well as assessing the models' effectiveness in the DRL parking environment. The results show that the BNN surrogate outperforms the MLP baseline in terms of practical utility for failure discovery. These findings suggest that BNNs can be a more effective surrogate model for prioritising failure scenarios in DRL testing.

1 Introduction

Reinforcement Learning (RL) has achieved remarkable success across various domains, including strategic game playing (e.g., AlphaGo), robotic control, and autonomous driving [1, 2]. These systems learn complex behaviours through trial-and-error interactions with simulated environments. The introduction of Deep Reinforcement Learning (DRL) extended RL by leveraging deep neural networks to operate in high-dimensional, continuous spaces, thereby expanding its applicability to more complex tasks [3]. As a result, DRL has enabled further progress in areas such as healthcare, personalisation systems, and autonomous control [4, 5]. A particularly relevant example is from Audi, which demonstrated that a DRL agent could successfully park a scaled-down vehicle using sensor feedback and learned policies [6].

While training such agents has been attempted extensively, the testing phase remains a major bottleneck because the current practice involves running the agents on a set of randomly generated configurations. These configurations are different environment setups or initial conditions that influence the agent's behaviour. This practice is typically computationally expensive and inefficient because it involves thousands of simulation runs to find rare but critical failure cases [7].

To mitigate this challenge, researchers have proposed using surrogate models, which are machine learning models trained to predict whether a given environment configuration will result in failure, without the need to execute the full simulation. Prior work, such as that of Biagiola et al. [3], demonstrated that Multi-Layer Perceptrons (MLPs) can approximate failures in DRL environments, enabling significant reductions in testing time. However, this

line of research has so far focused almost exclusively on MLPs, leaving open the question of whether other surrogate models could improve failure prediction.

Our approach builds upon the Indago tool proposed by Biagiola et al. [3]. It is a search-based testing framework that integrates a surrogate model into a Genetic Algorithm (GA). The GA is used to evolve environment configurations over several generations, with the surrogate estimating the failure likelihood and guiding the search toward areas that it deems more likely to fail. The surrogate serves as a proxy during agent execution, allowing the GA in Indago to explore high-risk configurations without full DRL rollouts [3].

Indago was initially used with an MLP as the surrogate model. In our work, we replace the MLP with a Bayesian Neural Network (BNN), a probabilistic model that represents weights as distributions rather than fixed values. This enables the surrogate to handle data imbalance better and generalise more effectively across uncertain inputs [8].

The following main research question guides the study:

> **RQ1:** *How do Bayesian Neural Networks compare to Multi-Layer Perceptrons as surrogate models for failure prediction in Deep Reinforcement Learning?*

This study is further decomposed into two sub-questions:

> **RQ1.1:** *How do their predictive performances compare, particularly under class imbalance?*

> **RQ1.2:** *How effectively can BNNs guide Genetic Algorithm-based test generation toward discovering new failure scenarios?*

Our findings show that BNNs can be more accurate and precise, but they show no significant improvement in metrics such as F1-score and AUC-ROC. On the other hand, our findings also show that, across 50 experiments in the Indago tool, the BNN-guided search discovered 27.8% more failure cases compared to the MLP baseline, along with significantly greater diversity in the types of failures uncovered.

This work makes the following contributions:

- We evaluate the ability of BNNs to classify failures based on environment configurations in a DRL testing scenario.
- We assess the effectiveness of BNNs as surrogate models for guiding a GA toward high-risk environment configurations.
- We empirically compare BNNs and MLPs in both predictive accuracy and their effectiveness in guiding failure discovery within a DRL testing pipeline.

2 Background and Motivation

2.1 Reinforcement Learning and Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a powerful learning paradigm in which agents learn optimal behaviours by interacting with an environment to maximise cumulative rewards [9]. It extends traditional Reinforcement Learning (RL) by integrating deep neural networks, which serve as powerful function approximators to model complex policies and value functions. This feature allows DRL agents to operate effectively in high-dimensional or

continuous state and action spaces where classical RL methods struggle. These capabilities have enabled successes in domains such as robotics, gaming, and autonomous vehicles [9, 2].

However, while training DRL agents has received extensive attention, the testing phase remains a significant challenge. In practice, testing a trained DRL policy involves executing it in thousands of simulated configurations to detect failure scenarios, which is computationally expensive [7]. As the agent becomes more competent, failures become increasingly rare; yet, these rare failures are precisely the most important to identify and address before deployment. Testing must therefore be both efficient and targeted toward high-risk scenarios, motivating the use of surrogate models [3].

2.2 Surrogate Modelling for Failure Prediction

A surrogate model approximates simulation outcomes by learning a mapping from environment configuration to failure likelihood. In DRL testing, this enables fast failure prediction without full simulation, reducing computational cost and enabling targeted search.

Previous work has explored the use of Multi-Layer Perceptrons (MLPs) as surrogate models in DRL testing pipelines. For example, the Indago tool by Biagiola et al. trained MLPs on interaction data to guide a Genetic Algorithm (GA) toward likely failure configurations [3]. However, MLPs exhibit a key limitation: they struggle with severe class imbalance, where failure cases constitute only a small fraction of the dataset. This often leads to models biased toward predicting the majority (non-failure) class [10]. To address this, we propose replacing MLPs with Bayesian Neural Networks (BNNs), which are more robust in data-scarce and imbalanced settings.

2.3 Bayesian Neural Networks

Bayesian Neural Networks extend standard neural networks by treating weights as probability distributions rather than point estimates, which can be visuallised in Figure 1. This allows them to model uncertainty in the network parameters, introducing a form of regularisation that improves robustness in sparse or imbalanced data settings [11, 12]. In the context of DRL surrogate modelling, these properties help reduce overfitting to the dominant (non-failure) class and enable better generalisation to rare failure cases.

Modern libraries such as BLiTZ (Blitz) [14] make scalable BNNs accessible in PyTorch [15], using variational inference to approximate the intractable posterior over network weights. Variational inference works by introducing a tractable distribution $q(w)$ over the

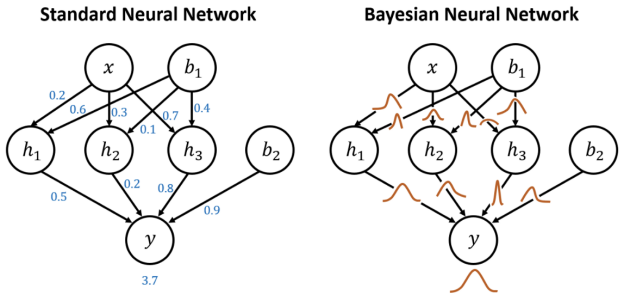

Figure 1: Adapted image [13] that shows the comparison between a standard Neural Network (left) and a Bayesian Neural Network (right).

weights w, and training the model to make $q(w)$ as close as possible to the true posterior $p(w \mid \mathcal{D})$, where $\mathcal{D}$ denotes the training data. In practice, this corresponds to minimising a loss derived from the Evidence Lower Bound (ELBO), which balances the likelihood of the data with a regularisation term, as shown in Equation (1):

$$\mathcal{L}_{\text{BNN}} = \underbrace{-\mathbb{E}_{q(w)}\left[\log p(\mathcal{D} \mid w)\right]}_{\text{Negative log-likelihood}} + \underbrace{\text{KL}\left(q(w) \,||\, p(w)\right)}_{\text{KL divergence regularisation}} \tag{1}$$

The first term, the *negative log-likelihood*, penalises the model when its predictions deviate from the observed labels. For binary classification tasks, this corresponds to the binary cross-entropy loss. The second term is the *Kullback-Leibler (KL) divergence*, which acts as a regulariser by penalising deviation of the variational distribution $q(w)$ from the prior distribution $p(w)$. This helps prevent overfitting and promotes generalisation in underrepresented regions of the input space.

We implement this objective in Blitz using variational Bayesian linear layers. These layers sample weights from $q(w)$ during forward passes. Combined with standard components like LogSoftmax [15] at the output, the resulting BNNs can be trained via stochastic gradient descent in a loop nearly identical to that of a standard PyTorch model. This makes BNN surrogates easy to integrate into the Indago tool with minimal code modifications, offering a practical and scalable choice for surrogate modelling in DRL testing pipelines.

2.4 Genetic Algorithms in Failure Discovery

Genetic Algorithms are a class of evolutionary optimisation techniques inspired by natural selection [9]. They operate by maintaining a population of candidate solutions (in our case, environment configurations) which evolve over successive generations based on principles of selection, crossover, and mutation. GAs are particularly suited to black-box optimisation problems where the search space is large, high-dimensional, or poorly understood, making them a compelling choice for DRL testing where analytical gradients are unavailable.

In the context of this project, we employ a GA as a search-based failure discovery mechanism. Instead of randomly sampling configurations to test the robustness of a DRL agent, the GA uses a surrogate model to assign each candidate a fitness score, defined as the predicted likelihood of failure. This prediction-driven evolution allows the algorithm to iteratively prioritise high-risk configurations, improving sample efficiency and reducing the number of full simulation calls required. Previous work such as Indago [3] has shown that combining surrogate-based fitness evaluation with evolutionary search can significantly accelerate the discovery of edge cases in safety-critical environments. Our work extends this idea by evaluating Bayesian surrogates as the predictive backbone of the GA loop.

2.5 Testing for Deep Reinforcement Learning Agents

Biagiola et al. [3] proposed a surrogate modelling approach for DRL testing, where an MLP is trained to classify failure outcomes given environment configurations. Their Indago framework combines this surrogate with a search-based failure discovery mechanism using a GA, reducing the need for random sampling by iteratively evolving high-risk configurations based on predicted failure likelihood, rather than sampling blindly from the entire configuration space.

Beyond this, surrogate models have been applied to other reinforcement learning tasks, such as selecting representative DRL trajectories that prioritise behavioural diversity and certainty [16], and replicating DRL behaviour in Atari environments [17]. However, their

role in failure prediction, especially in simulation-heavy DRL environments, remains underexplored.

In parallel, research in uncertainty estimation has received increasing attention. Kendall and Gal [8] highlighted the role of epistemic uncertainty in improving model reliability, particularly in safety-critical tasks. Malinin and Gales [10] demonstrated that uncertainty-aware models are better calibrated, especially in imbalanced classification tasks. BNNs, in particular, have been studied for their ability to capture epistemic uncertainty through variational inference [11], but to our knowledge, they have not yet been applied to surrogate failure prediction in DRL. This work builds on these ideas by evaluating BNNs as replacements for MLPs within the Indago framework.

3 Approach

The goal of this project is to reduce the computational cost of testing Deep Reinforcement Learning (DRL) agents by building a surrogate model that can predict whether a given environment configuration is likely to result in a failure (collision) without executing the full simulation. This section presents our proposed solution for improving the efficiency and effectiveness of DRL agent testing using surrogate modelling, which we evaluate in comparison to an MLP-based surrogate.

3.1 Case Study: Parking Environment

As a concrete case study for evaluating our surrogate model, we focus on the *Parking environment* from the HighwayEnv simulator [18]. In this scenario, a DRL agent controls a car known as the *ego* vehicle, a term commonly used to refer to the autonomous vehicle being controlled in a simulation. The agent must navigate the car to a designated lane and park it without colliding with other vehicles. Actions include continuous throttle and steering, and each episode terminates upon success or failure. For the purposes of this study, a failure is defined as an episode that ends in either a collision or a timeout, indicating that the agent was unable to complete the parking task successfully. Figure 2 illustrates a possible configuration. Subfigure (A) shows the simulated layout with ego and parked vehicles, while (B) presents the JSON-style configuration used as input to the surrogate.

Each test case is encoded as a feature vector comprising:

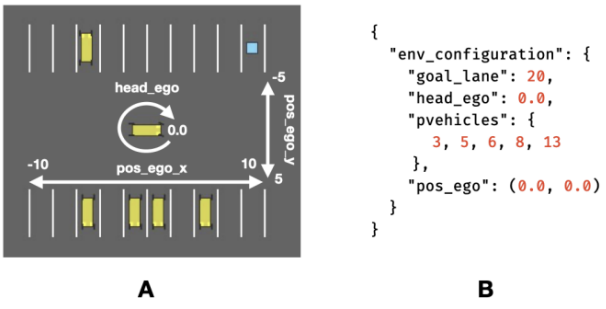

Figure 2: Figure adapted from Biagiola et al. [3], showing an example configuration in the Parking environment. (A) Simulation state with ego vehicle, parked cars, and goal lane. (B) JSON-style configuration used as surrogate model input.

- The ego vehicle's initial `(x, y)` position
- Its heading angle `head_ego`
- The target parking lane `goal_lane` (1–20)
- A binary occupancy vector `pvehicles` over 20 lanes

These input features are flattened into a fixed-length vector suitable for model input. The `pvehicles` field, which indicates which of the 20 parking lanes are occupied, is represented using one-hot encoding, a binary format where each lane is assigned a bit set to 1 if occupied and 0 if free. This is followed by the goal lane index, the heading angle of the ego vehicle, and its `(x, y)` position.

3.2 Data Collection and Preprocessing

The dataset used in this work was originally generated by Biagiola et al. [3] as part of their DRL agent training experiments in the Parking environment. It consists of 8,790 data points collected from agent-environment interactions during learning with a DRL agent trained using Hindsight Experience Replay (HER) [19]. Each data point includes the environment configuration introduced in subsection 3.1, along with the actions taken by the agent and the outcome (success or failure). In our surrogate model training, we use only the environment configuration as input features, since this is the only information available prior to simulation in the DRL environment. The binary success/failure outcome is used as the target label.

We also consider the temporal nature of data collection: earlier data points were generated when the agent was less trained, whereas later ones reflect a more competent agent. For this reason, we primarily use the latter portion of the dataset, specified by the *training progress filter* parameter. This filter excludes an initial percentage of data points, as early-stage trajectories reflect unrealistic agent behaviour unlikely to occur once the policy is fully trained. This parameter was set to 50, as done by Biagiola et al. [3].

3.3 Surrogate Modeling Workflow

Our surrogate modelling workflow consists of five sequential steps: data preprocessing, model development, hyperparameter tuning, search-based failure prioritisation, and evaluation.

We begin by preprocessing the dataset as described in subsection 3.2, and split it into training and test sets using an 80/20 ratio, a division that has been empirically shown to offer strong generalisation performance [20].

Next, we develop the surrogate model using a Bayesian Neural Network (BNN) implemented with the BLiTZ library [14]. The network is built in PyTorch using a sequence of Bayesian linear layers (`BayesianLinear`), which replace standard deterministic `Linear` layers. These layers treat each weight and bias as a Gaussian distribution parameterised by a mean and standard deviation, which were initialised with a prior of $\mathcal{N}(0, 1)$, enabling the model to learn a distribution over weights rather than point estimates. During each forward pass, weights are sampled from these learned distributions, resulting in stochastic predictions. The final layer of the BNN is a `LogSoftmax` output, and training is performed by minimising the ELBO loss as described in Section 2.5.

The rest of the BNN architecture follows a standard feedforward design with tunable parameters, including the number of hidden layers and hidden layer size. We use ReLU activations between layers and train all models using the Adam optimiser. Classification is framed as a binary task where the BNN outputs log-probabilities for success and failure. At

inference time, we exponentiate the log-probabilities to obtain predicted class probabilities and apply a fixed threshold of 0.5 to assign binary labels.

To address the severe class imbalance, where failure cases represent only 2% of the data, we apply several mitigation strategies. *Oversampling* duplicates failure samples to increase their relative frequency in the training batches, while *undersampling* randomly discards non-failure samples to rebalance the dataset; both are controlled via a tunable ratio parameter. *Class-weighted loss* penalises misclassifications of the minority class more heavily, allowing the model to learn a more balanced decision boundary. Additionally, we use targeted *data augmentation* that perturbs failure configurations slightly to generate synthetic samples near the failure regions, increasing diversity while maintaining physical plausibility. We did not use other techniques like SMOTE [21], as it creates synthetic minority examples by interpolating between existing failure cases. However, in our structured input space, even if two configurations result in failure, their interpolated midpoint does not necessarily represent a failure scenario.

Following initial development, we fine-tuned the BNN architecture and training setup via grid search, systematically exploring combinations of hyperparameters such as hidden layers, layer sizes, and imbalance-handling strategies (oversampling, undersampling, data augmentation, class-weighted loss). Each configuration was first ranked using validation AUC-ROC, a threshold-independent metric suited for imbalanced datasets as it captures class separability across all thresholds [22]. A classification threshold is the cutoff used to decide whether a predicted probability (e.g., 0.7) is considered a positive or negative class. AUC-ROC evaluates model performance across all such thresholds rather than relying on one fixed cutoff. Final evaluation was performed on a held-out test set using accuracy, precision, recall, F1-score, and AUC-ROC. Accuracy reflects overall correctness, precision measures the proportion of predicted failures that were correct, recall indicates how many actual failures were detected, and F1-score balances both [22]. Precision, recall, and F1-score were computed using a 0.5 classification threshold, consistent with the original setup in prior work [3].

The next step is integrating the model into a failure search loop powered by a Genetic Algorithm (GA). In this phase, the surrogate model replaces the DRL agent and is used to compute the fitness value of each candidate configuration, defined as the predicted probability of failure. These fitness scores are then used to guide the GA's evolutionary process, allowing it to prioritise and evolve high-risk configurations without executing full DRL simulations. The pseudocode of the GA procedure is shown in Appendix B

Finally, we evaluate the model's performance within this GA setting, measuring the average number of failures discovered and analysing the diversity of the failures found, and compare our results against the Multi-Layer Perceptron (MLP) baseline from Biagiola et al. [3].

4 Study Design

4.1 Research Questions

The following main research question guides the study:

> **RQ1:** *How do Bayesian Neural Networks compare to Multi-Layer Perceptrons as surrogate models for failure prediction in Deep Reinforcement Learning?*

This study is further decomposed into two sub-questions:

RQ1.1: *What is the performance of BNN compared to the MLP in classifying failing environments?*

RQ1.2: *How effectively can BNNs, compared to the MLP, guide Genetic Algorithm-based test generation toward discovering new failure scenarios?*

4.2 Evaluation Criteria

4.2.1 Baseline Model

Our baseline is the Multi-Layer Perceptron (MLP) surrogate model originally used by Biagiola et al. [3] in the Indago framework. It consists of a feedforward architecture with two hidden layers and ReLU activations, trained using binary cross-entropy loss. Rather than retraining this model, we obtained the original trained checkpoint (`.pkl` file) directly from the authors, ensuring consistency with prior work.

4.2.2 Methodology

To evaluate classification performance, we trained BNN models to predict whether a given Parking environment configuration would result in a failure. We conducted a grid search over 144 BNN configurations, derived from a factorial combination of the following hyperparameters:

- Number of hidden layers (1-4)
- Hidden layer size (32, 64, 128)
- Oversampling/Undersampling ratio (0.0, 0.5, 1.0)
- Use of undersampling or oversampling (boolean)
- Use of data augmentation (boolean)
- Use of class-weighted loss (boolean)

This search space was constrained by computational budget, given the cost of training BNNs and the need to average results across multiple seeds. While not exhaustive, the 144 configurations span a representative range of settings for architecture and imbalance handling. Each configuration was trained using five random seeds and evaluated on a fixed test set. Performance was assessed using accuracy, precision, recall, F1-score, and AUC-ROC. Early stopping was applied based on validation loss. All reported metrics represent the mean across seeds.

To complement the comparison of classification metrics between the BNN and MLP, we conducted a statistical significance analysis using the *Mann-Whitney U test*, a non-parametric test used to assess whether one distribution tends to produce higher values than another without assuming normality [23]. We also report the one-sided *Vargha-Delaney* $\hat{A}_{12}$ effect size, which quantifies the probability that a randomly chosen value from one distribution exceeds one from another [24]. This analysis was applied to determine whether the BNN achieved significantly higher values in key metrics, supporting more robust conclusions about relative model performance.

To evaluate the practical utility of BNN surrogates in a DRL testing pipeline, we integrated each model into a Genetic Algorithm (GA) designed to generate and evolve candidate

environment configurations. We selected the top 10 BNN configurations from the gridsearch based on validation AUC-ROC and ran each of them in the GA pipeline across three independent experiments of 50 episodes. This experimental budget was chosen to strike a balance between computational feasibility and the ability to observe consistent trends. Effectiveness was measured by the average number of failures discovered, as defined in subsection 3.1.

Subsequently, the BNN surrogate with the highest average number of failures was selected for further evaluation. This model was executed in 50 experiments using the GA, each running for 50 generations, which enabled a more stable estimate of performance and downstream robustness.

To complement the failure discovery results, we evaluated the diversity of the failures uncovered by the GA. This analysis considers between two dimensions: **input diversity**, which measures variation in the initial environment configurations (e.g., ego position, goal lane), and **output diversity**, which captures the variation in the types of failures produced. For each dimension, we report two metrics: **coverage**, which quantifies the proportion of the configuration or failure space explored, and **entropy**, which measures how evenly failures are distributed across that space [3]. Together, these metrics assess whether the GA explores a wide and meaningful range of scenarios or tends to converge on a limited subset of failures.

The BNN's performance in this extended evaluation was compared against a fixed MLP baseline, enabling an empirical comparison of their respective capabilities as surrogate models within a search-based DRL testing framework. To determine the statistical significance of observed performance differences, we adopted the same methodology as Biagiola et al. [3], applying the Mann-Whitney U test for significance testing and the Vargha-Delaney $\hat{A}_{12}$ effect size to quantify the magnitude of the difference between methods.

5 Results

5.1 RQ1.1: Classification Performance

Table 1 presents the top 10 configurations, reporting performance on the test set with 95% confidence intervals averaged over five random seeds.

While selecting the single best-performing model might seem optimal, prior work (e.g., Bergstra and Bengio) highlights the variability and uncertainty inherent in hyperparameter tuning [25]. When multiple configurations yield similarly high validation scores, it is common practice to analyse a top-k subset (in our case, top-10) to ensure robustness to noise and overfitting on the validation set. This also allows us to evaluate consistency across multiple seeds and better capture the model's general performance trend.

The boxplot analysis in Figure 3 reveals distinct performance trade-offs between the BNN and MLP models. BNNs tend to achieve higher precision and slightly better F1-scores, with greater variability across runs and configurations. In contrast, the MLP outperforms the BNN in recall and shows slightly better AUC-ROC scores, with notably more stable performance.

While the highest-ranked BNN based on validation AUC-ROC achieved strong recall, it underperformed the MLP baseline on several test metrics, including F1-score and AUC-ROC. In contrast, the seventh-best BNN configuration demonstrated more robust test performance, outperforming the MLP in accuracy, F1-score, and AUC-ROC. Given the fact that this configuration also achieved the best performance in the failure discovery task, we selected it for direct comparison with the MLP baseline. The full list of hyperparameters used in this configuration is provided in Appendix A.Table 2 summarises the results.

Table 1: Top 10 BNN configurations obtained in the grid search ranked by validation AUC-ROC (averaged over five seeds), using a fixed learning rate (0.01) and a fixed data split (0.1). Results include 95% confidence intervals. The selected configuration is shaded, and values exceeding the MLP baseline (Accuracy: 0.784, F1: 0.178, AUC-ROC: 0.697) are highlighted in bold.

Layers	Hidden	Under	Oversample	Augment	Weight Loss	Accuracy	F1	Test AUC-ROC
1	64	True	0.0	False	True	0.704 ± 0.123	0.159 ± 0.017	0.692 ± 0.009
1	64	False	0.0	False	True	0.704 ± 0.123	0.159 ± 0.017	0.692 ± 0.009
1	32	False	0.0	False	True	0.661 ± 0.060	0.153 ± 0.012	0.693 ± 0.004
1	32	True	0.0	False	True	0.661 ± 0.060	0.153 ± 0.012	0.693 ± 0.004
1	128	False	0.5	True	True	**0.866** ± 0.074	**0.220** ± 0.029	**0.709** ± 0.011
1	32	False	0.5	True	True	**0.840** ± 0.054	**0.191** ± 0.035	**0.701** ± 0.011
1	64	False	0.5	True	True	**0.877** ± 0.014	**0.191** ± 0.034	**0.703** ± 0.011
2	32	True	0.0	False	True	0.681 ± 0.298	0.160 ± 0.037	0.661 ± 0.046
2	32	False	0.0	False	True	0.681 ± 0.298	0.160 ± 0.037	0.661 ± 0.046
1	32	False	0.0	True	True	**0.848** ± 0.037	0.157 ± 0.016	0.682 ± 0.010

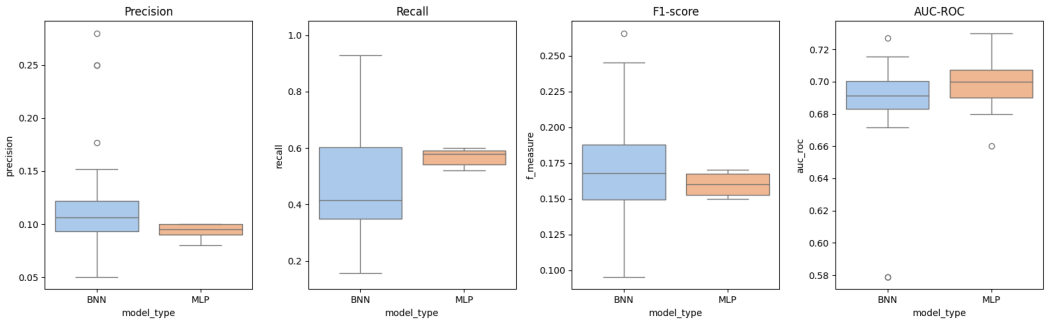

Figure 3: Distribution of precision, recall, F1-score, and AUC-ROC for the top BNN configurations on the left (our model) and ten MLP runs on the right (the baseline).

The BNN showed statistically significant improvements in **accuracy** ($p = 0.004$, $\hat{A}_{12} = 1.00$) and **precision** ($p = 0.005$, $\hat{A}_{12} = 1.00$). For **F1-score**, the BNN outperformed the MLP with a borderline effect ($p = 0.069$, $\hat{A}_{12} = 0.80$). However, in terms of **recall**, the MLP performed better, though the difference was not statistically significant ($p = 1.000$, $\hat{A}_{12} = 0.00$). **AUC-ROC** scores were comparable ($p = 0.500$, $\hat{A}_{12} = 0.52$). These findings reiterate the conclusions drawn from Figure 3 and suggest that while BNNs can significantly improve some metrics, their gains may come at the cost of recall. Nonetheless, the improvement in precision aligns with the priorities identified in prior work for efficient failure discovery [3].

5.2 RQ1.2: Failure Discovery via Genetic Algorithm

Table 3 presents the performance of the top 10 BNN configurations in guiding the Genetic Algorithm (GA) to discover failure cases, sorted by the average number of failures discovered over three runs.

The best-performing BNN model achieved an average failure discovery of 18.2. This model was further tested over 50 runs of 50 episodes each. The results are summarised in Table 4

The results in Table 4 report the performance and diversity metrics for the MLP and BNN surrogates across 50 runs of the GA. In terms of effectiveness, the BNN-guided GA identified a greater number of failing environments on average (19.14 vs. 14.98). This

Table 2: Comparison of results on the test set between MLP baseline and selected BNN (7th in validation AUC, 1-layer, 64-hidden, augmented, class-weighted). Bold indicates best result per metric. Significance tests were applied using one-sided Mann-Whitney U and Vargha-Delaney $\hat{A}_{12}$.

Metric	**MLP**	**BNN (Selected)**	**p-value**	$\hat{A}_{12}$
Accuracy	0.784 ± 0.040	**0.877** ± 0.014	**0.004**	**1.00**
Precision	0.110 ± 0.023	**0.191** ± 0.034	**0.005**	**1.00**
Recall	**0.477** ± 0.043	0.306 ± 0.071	1.000	0.00
F1-Score	0.178 ± 0.031	**0.191** ± 0.034	0.069	0.80
AUC-ROC	0.697 ± 0.028	**0.703** ± 0.011	0.500	0.52

Table 3: Performance of the top 10 BNN configurations in the Genetic Algorithm, ranked by average number of failures discovered across three runs of 50 episodes.

Layers	**Hidden**	**Under**	**Oversample**	**Augment**	**Weight Loss**	**Failures Discovered**
1	64	False	0.5	True	True	18.2
1	64	False	0.0	False	True	16.8
1	32	False	0.0	True	True	13.2
1	128	False	0.5	True	True	13.0
1	32	True	0.0	False	True	11.3
1	32	False	0.0	False	True	10.8
1	64	True	0.0	False	True	10.0
2	32	False	0.0	False	True	9.5
2	32	True	0.0	False	True	9.3
1	32	False	0.5	True	True	8.3

difference was statistically significant, with a p-value of 1.45×10^{-7} and a Vargha-Delaney effect size of 0.804, indicating a *large* effect favouring the BNN.

The diversity of input configurations was comparable between models, as expected, since both models receive the same environment configuration features as input within the GA. However, the BNN achieved significantly greater output diversity. In particular, its output coverage (75.64% vs. 43.36%) and entropy (50.68 vs. 22.06) were both significantly higher, with p-values of 6.63×10^{-15} and 5.04×10^{-6}, and effect sizes of 0.936 and 0.758 respectively, both indicating *large* effects in favour of the BNN.

These results confirm that the BNN surrogate not only discovered more failures but also a broader and more varied set of failure behaviours. Higher output diversity implies

Table 4: Comparison of diversity and performance metrics between the MLP baseline and BNN across 50 GA runs where bold means statistically significantly better. Input diversity is measured by coverage and entropy of input configurations; output diversity is measured from the coverage and entropy of failure outputs. Statistical significance was determined using Mann-Whitney U tests with $p < 0.05$.

Category	**Metric**	**MLP**	**BNN**
Performance	Failing Environments	14.98 ± 3.24	**19.14** ± 3.75
Input Diversity	Coverage (%)	50.00	52.00
	Entropy	0.00	1.17
Output Diversity	Coverage (%)	43.36	**75.64**
	Entropy	22.06	**50.68**

that the BNN-guided GA explored a wider range of distinct failure scenarios, increasing the likelihood of uncovering rare or unexpected edge cases that a less exploratory model might miss.

6 Threats to Validity

Construct Validity We define failure as either a collision or a timeout, both of which are critical and unambiguous events in the Parking environment. Our evaluation uses F1-score and AUC-ROC, which are well-suited to binary classification tasks.

Internal Validity Our Multi-Layer Perceptron (MLP) baseline uses the configuration reported by Biagiola et al. [3], assuming its optimality holds in our adapted setup. All models use identical train-test data splits and preprocessing to ensure fair comparison. We averaged results across multiple random seeds and used early stopping to prevent overfitting. We conducted a grid search over several architectural and imbalance-related hyperparameters, other parameters such as the learning rate, batch size, and optimiser were fixed and not further tuned, which may limit the performance potential of the BNNs.

External Validity Our model is trained solely on the Parking environment, which may limit generalisability to other Deep Reinforcement Learning (DRL) tasks. However, the core methodology of training BNN surrogates on structured environment configurations remains transferable.

Conclusion Validity We mitigate evaluation randomness by averaging results over multiple random seeds and report 95% confidence intervals. For RQ1.1 and RQ1.2, we conduct statistical significance testing using the Mann-Whitney U test and Vargha-Delaney effect size. However, as our comparisons are limited to a single environment, generalising these results beyond the studied environment should be done cautiously.

7 Conclusion, Limitations and Future Work

This study set out to compare the effectiveness of Bayesian Neural Networks (BNNs) and standard Multi-Layer Perceptrons (MLPs) as surrogate models for failure prediction in Deep Reinforcement Learning (DRL) testing. Prior research relied exclusively on MLP surrogates and performed minimal hyperparameter tuning, raising the question of whether other machine learning models could perform better in this task. To this end, we developed and evaluated a suite of BNN surrogates trained on environment configurations, applying them to guide a Genetic Algorithm (GA) in prioritised failure discovery.

We found that BNNs could achieve higher metrics, such as precision and F1-score, compared to the MLP. Furthermore, the BNN surrogate significantly outperformed the MLP in a GA-guided failure discovery task, discovering a greater number and greater diversity of failures. These results suggest that probabilistic surrogates can enhance the practical utility of DRL testing pipelines.

Limitations Firstly, the evaluation was restricted to a single environment (Parking), which limits generalisability to other DRL settings. Secondly, our choice of BNN framework (Blitz) offered practical convenience but may underutilise the theoretical strengths of Bayesian inference.

Future Work Future directions include exploring alternative probabilistic models, such as Deep Ensembles or Pyro-based BNNs, for more expressive uncertainty modelling. The surrogate training process could also benefit from a larger or more advanced hyperparameter search, such as Bayesian optimisation. Finally, applying the surrogate modelling pipeline to additional DRL environments would help assess the generalisability of the proposed approach.

In summary, our findings provide empirical evidence that BNNs can be a viable alternative to MLPs for surrogate-based failure prediction in DRL, at least within the Parking environment studied. While not a definitive solution, they showed promising improvements in failure discovery effectiveness and suggest potential benefits for testing approaches in reinforcement learning.

8 Responsible Research

This project aims to improve the safety and efficiency of testing in DRL environments by developing a surrogate model for predicting agent failures. While the model is not deployed in a real-world setting, responsible research practices were followed throughout.

8.1 Ethical Considerations

The surrogate is designed to reduce computational cost by identifying high-risk configurations before simulation. However, surrogate predictions must be interpreted cautiously in safety-critical contexts. We evaluate model performance against ground-truth outcomes from full DRL simulations to mitigate this.

All data used in this study were collected from a simulated environment, the Parking scenario within the HighwayEnv framework, meaning no real-world agents or humans were involved. All models were trained and tested on consistent data splits with identical preprocessing and evaluation metrics. To ensure robustness, each experiment was repeated over five random seeds, and performance is reported as mean scores across runs.

8.2 Reproducibility

All experiments were run on a fixed dataset of 8,790 configurations generated in the Parking environment. Preprocessing steps, training protocols, and hyperparameter settings are explicitly recorded. The experimental pipeline, including data splits, model definitions, and search configurations, has been followed as documented in section 3.

To account for randomness in model training, we followed the same evaluation practice as Biagiola et al. [3], using random seeding across five runs for each configuration and reporting the mean and 95% confidence intervals. The five seeds used in this research are as follows: 21, 22, 23, 24, and 25.

The simulation framework (HighwayEnv), surrogate model (PyTorch + Blitz), and required dependencies are publicly available or installable via standard package managers. The full source code for reproducing all experiments and figures is available at: `https://github.com/rodrigo-montero/surrogate-reloaded`.

8.3 Use of AI

Large Language Models (LLMs) were used during the research process to assist with non-substantive tasks. Specifically, they were used to create the initial LaTeX table skeleton for

Table 4, to support the search and summarisation of relevant academic literature, and to help format the equation presented in subsection 2.5 in LaTeX. Additionally, Grammarly Premium was used for grammar and clarity checks; this tool may use AI for some of its suggestions.

8.4 System Setup

Experiments were conducted on a MacBook Pro (Intel CPU, 16GB RAM) without a dedicated GPU. Full grid searches required between 5-24 hours, depending on the configuration space. The only additional dependency was the `blitz-bayesian-pytorch` package.

8.5 Acknowledgements

I would like to thank my professor, supervisor, and fellow research peers for their valuable guidance, feedback, and collaboration throughout this project.

References

[1] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[2] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day, 2018.

[3] 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.

[4] Matthieu Komorowski, Leo A. Celi, Omar Badawi, Anthony C. Gordon, and A. Aldo Faisal. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature Medicine*, 24(11):1716–1720, October 2018.

[5] Netflix Technology Blog. Artwork personalization at netflix - netflix techblog. *Medium*, June 2018.

[6] Webmaster. Automatic intelligent parking: Audi at nips in barcelona. *Automotive World*, December 2016.

[7] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, Mojtaba Bagherzadeh, and Ramesh S. A search-based testing approach for deep reinforcement learning agents. *IEEE Transactions on Software Engineering*, 49(7):3715–3735, July 2023.

[8] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision?, 2017.

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second edition, in progress edition, 2014.

[10] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks, 2018.

[11] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks, 2015.

[12] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR.

[13] Gabriel Costa. A first insight into bayesian neural networks (bnns). *Medium*, December 2022.

[14] Pi Esposito, Jonas Fill, Daniel Kelshaw, Hannan4252, Ana Tamais, Ana Tamais, Anders U. Waldeland, Kobi Felton, Lucas Kruitwagen, Michal Karbownik, piotr-rarus sh, Sarthak Pati, Kirill, and sansiro77. *piEsposito/blitz-bayesian-deep-learning*. 9 2023.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

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

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

[18] Edouard Leurent. An environment for autonomous driving decision-making. *GitHub repository*, 2019. `https://github.com/eleurent/highway-env`.

[19] 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.

[20] 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.

[21] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[22] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. volume 06, 06 2006.

[23] Henry B. Mann and Douglas R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60, 1947.

[24] 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.

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

A Final Bayesian Neural Network Hyperparameters

To ensure reproducibility, Table 5 details the full set of hyperparameters used in the final BNN model configuration, selected after grid search and robustness analysis.

Table 5: Final hyperparameter values used in the best-performing BNN surrogate model.

Category	**Value**
Number of hidden layers	1
Hidden layer size	64
Activation function	LogSoftmax
Optimizer	Adam
Learning rate	0.001
Batch size	128
Early stopping	Based on validation loss
Patience	10
Test split	0.2
Validation split	0.1
Epochs	40
Training Progress Filter	50
DRL algorithm	her
Oversampling ratio	0.5
Undersampling	False
Class-weighted loss	True
Data augmentation	True
Bayesian framework	Blitz for PyTorch
Distribution Priors	Gaussian (Blitz default)
Seeds used	[21, 22, 23, 24, 25]

B Genetic Algorithm Pseudocode

The pseudocode in Algorithm 1 outlines the Genetic Algorithm used to discover failing environment configurations. The primary modification introduced in this work is the choice of the classifier f, which serves as the fitness function during the search. The rest of the parameters are kept the same to ensure consistency with prior research.

Algorithm 1: Genetic algorithm for the generation of environment configurations

Input: f, classifier;
PS, population size;
cr, crossover rate;
E_f, set of environment configurations in which the DRL agent failed during training
Output: $\hat{e}$, environment configuration to execute the agent on

```
Generate the initial population of environment configurations and compute the corresponding fitness
population ← GeneratePopulation(PS, E_f)
ComputeFitness(population, f)
currentIteration ← 0
Main loop that changes the population guided by the classifier f until the timeout expires
repeat
    Build the new population by extracting a certain percentage of the best environment configurations
    newPop ← Elitism(population)
    Fill the rest of the population by evolving the environment configurations
    while |newPop| < PS do
        Select the best environment configurations according to their fitness
        pe_1 ← Selection(population)
        pe_2 ← Selection(population)
        Copy the environment configurations (offspring) to avoid changing the original ones (parents)
        oe_1 ← Copy(pe_1)
        oe_2 ← Copy(pe_1)
        Crossover two environment configurations with a certain probability cr, ensuring their validity
        if GetRandomFloat() < cr then
            oe_1, oe_2 ← Crossover(oe_1, oe_2)
        Mutate the offsprings ensuring their validity
        oe_1 ← MutateEnvConfig(oe_1)
        oe_2 ← MutateEnvConfig(oe_2)
        /Add the best environment configurations to the population according to their fitness
        AddBestIndividuals(newPop, pe_1, pe_2, oe_1, oe_2)
    Compute the fitness of environment configurations in the new population
    population ← newPop
    ComputeFitness(population, f)
    Replace the worst environment configurations in the population to avoid stagnation
    ReseedPopulation(population, currentIteration, E_f)
    currentIteration ← currentIteration + 1
until ¬ timeout();
Extract the environment configuration with the best fitness
ê ← GetIndividualWithBestFitness(population, f)
return ê
```