# Surrogate Reloaded: Fast Testing for Deep Reinforcement Learning.

## Convolutional Neural Networks as surrogate model for DRL testing

**Leon Braszczyński**

**Supervisor(s): Dr. A. Panichella, 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: Leon Braszczyński
Final project course: CSE3000 Research Project
Thesis committee: Annibale Panichella, Antony Bartlett, Cynthia Liem

An electronic version of this thesis is available at http://repository.tudelft.nl/.

# Surrogate Reloaded: Fast Testing for Deep Reinforcement Learning

Convolutional Neural Networks as surrogate model for DRL testing


## Abstract

In recent years, Deep Reinforcement Learning (DRL) has moved away from playing games to more practical tasks like autonomous parking. This transition has created a need for efficient testing of DRL agents. To evaluate an agent, we need to run a simulation of the task and let the agent decide what actions to perform. Running these simulations is expensive and time-consuming. The problem of testing is that we need hundreds of different test scenarios, and each simulation takes from seconds to minutes depending on the use case, which is why choosing the right initial state of the simulation is critical. Current solutions leverage the idea of surrogate models that can approximate how difficult it will be for an agent to complete a given environment without running the tests. Existing work has explored the use of surrogate models for DRL tasks, creating a Multi-Layer Perceptron to act as a surrogate model for a DRL agent attempting to park a car in a parking lot. Parking scenarios are inherently spatial problems, and MLPs are not able to take advantage of that spatial structure. To address this limitation, we used Convolutional Neural Networks (CNNs), which are designed to handle spatial information more effectively, and should therefore improve prediction performance over MLPs. Our approach transforms tabular data into a low-resolution grid-like image representing a parking lot. This approach guides a genetic algorithm to discover a mean of 25.76 failures per run, a 72% improvement over the 14.98 failures found by the MLP baseline. Our model also achieves significantly higher scores on diversity metrics of generated environments.


## 1 Introduction

Deep Reinforcement Learning (DRL) is currently used across many fields. It is used by Netflix for movie recommendations and by autonomous vehicle companies like Waymo [13] [11]. DRL agents learn how to take the right decisions not from examples (data) like most ML models, but by learning a policy, which is a mapping from states to actions [27]. They achieve this by receiving rewards for interacting with a simulated environment. Through trial and error, agents learn without need for supervision. In essence, they learn optimal decisions by engaging with an environment where correct decisions are rewarded and wrong decisions are penalized. For example, in a parking scenario, an agent is rewarded for parking a car correctly and punished for hitting other cars. By interacting with the environment long enough, DRL agents can learn what the right decision is depending on the state they are in.

DRL agents can significantly impact human health and safety, especially if they are used in safety-critical systems like autonomous driving [1]. Therefore, the demand for testing DRL agents is growing [1]. To ensure their safe and reliable deployment in real-world applications, rigorous and advanced testing methodologies are essential.

To evaluate a trained agent, it is redeployed in the simulation environment used during training. However, rather than relying on reward signals, the agent's performance is assessed based on its ability to complete the intended task [21]. We can check if the agent was able to park the car correctly, or if it failed and crashed into any obstacles. We can measure and evaluate the agent on multiple metrics, but this is insufficient. We must also consider the difficulty of the environment, features such as the size of the parking lot, the distance from the target parking spot, and the number of obstacles present as well as their placement, influence the agent's performance during testing. In testing, we focus on edge cases and scenarios that are more challenging than normal conditions. We aim to generate scenarios that are as diverse and difficult as possible. The brute-force method of generating thousands of random scenarios is computationally expensive, as running each simulation can take about 40 seconds, even reaching up to 5 minutes in some cases. Therefore, developing an efficient method to generate challenging scenarios is essential.

To address the issue of running computationally expensive DRL models, Biagiola and Tonella proposed the usage of surrogate models as a proxy for DRL agents to estimate the outcome of simulations without executing them [3]. Testing DRL agents requires hundreds of different test scenarios for in-depth evaluation, and each scenario is expensive to run. Hence the motivation for using surrogate models. A surrogate model is significantly more efficient than a DRL agent, allowing us to evaluate the difficulty of multiple environments without needing to run expensive simulations. This, combined with a genetic algorithm, provides a method to generate environments that are more diverse and have a higher probability of failure than traditional approaches.

Their work was limited, as they only used a single multilayer perceptron (MLP) surrogate that didn't undergo extensive hyper-parameter tuning. This represents a missed opportunity, as autonomous driving problems are inherently spatial problems. MLPs flatten all spatial information before processing, thus potentially losing insight. We conjecture that this limitation impacts the effectiveness of the surrogate model.

Therefore, in this work, we implement an additional surrogate using Convolutional Neural Networks (CNNs). Prior research has shown that CNNs work well with spatial data [12]; they can understand spatial patterns more effectively. As an analogy, it is easier for humans to look at a visualization of a parking lot and decide if it represents a simple or difficult scenario than to interpret the tabular data of the environment. CNNs are also known for their translational invariance [12], ensuring that challenging features can be detected regardless of their position in the input. Based on this reasoning, we develop CNN-based surrogate models. We also perform extensive hyper-parameter and architecture fine-tuning of a CNN model using grid search [15].

The model relies on transformed data. We transformed tabular data used by the original MLP into a low-resolution image representation. It is a 3-channel grid of size 4x10. A detailed description of the transformation can be found in Methodology Section.

We evaluate our approach using the perpendicular parking task from the HighwayEnv simulator [14], where a DRL agent must park a vehicle in a designated spot while avoiding collisions with other cars. To answer our research questions, we develop an optimized CNN model and perform extensive grid search to tune hyperparameters and architecture. We evaluated classification performance of our CNN using six evaluation criteria: precision, recall, F1-score, accuracy, AUC-ROC and validation loss [5]. We use the best-performing CNN configuration to guide a Genetic Algorithm (GA) to find environment configurations which the DRL agent is likely to fail on and compare their performance using three evaluation criteria: mean probability of generating the failing environment, coverage, and entropy.

Our experiments demonstrate that the CNN-based surrogate model significantly outperforms the MLP baseline. The CNN-guided search discovered a mean of 25.76 failures per run, a 72% improvement over the MLP's 14.98 failures. The advantage extended to all diversity metrics: for example, in the input diversity category, the CNN achieved 70.0% coverage and 19.6% entropy, substantially outperforming the MLP's 59.0% coverage and 6.8% entropy. These results indicate a more comprehensive and varied test case generation.

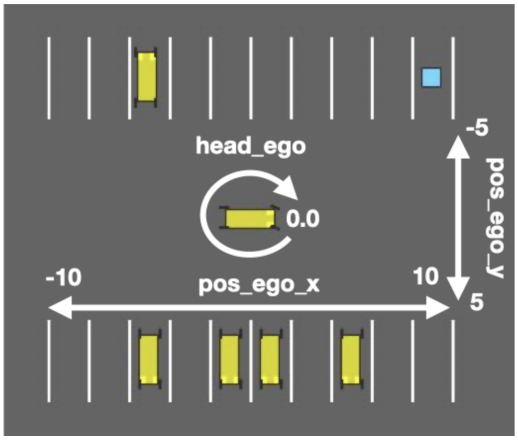
This paper aims to answer the following research questions:

- **RQ1:** What architectural and design choices lead to effective CNN-based surrogate models for DRL testing?
- **RQ2:** How does the performance of the CNN-based surrogate model compare to an MLP-based surrogate baseline?

To answer these questions, our main contributions are as follows:

- We evaluate the performance of CNNs in classifying failing environment configurations for the parking task.
- We demonstrate the effectiveness of using CNNs to guide a GA in finding these failing configurations.
- We provide a comprehensive comparison of the overall performance of CNN and MLP surrogate models for testing DRL agents.
- We introduce a data transformation approach that converts tabular environment data into a spatial representation suitable for CNN processing.
- We identify and validate a more reliable design choice for surrogate model selection, showing that a model's generalization performance is a better predictor of success than standard classification metrics.

The remainder of this paper is organized as follows. Section 2 provides the necessary background, while Section 3 describes our methodology for transforming the data, developing, and training the CNN. Section 4 details the implementation, and Section 5 outlines our study design. We present our results in Section 6 and discuss threats to validity in Section 7. Finally, Section 8 offers a conclusion and recommendations for future work, and Section 9 discusses ethical considerations and reproducibility.




(a)


```
{
  "env_configuration": {
    "num_lanes": 10,
    "goal_lane_idx": 20,
    "head_ego": 0.0,
    "pvehicles": {
      3, 5, 6, 8, 13
    },
    "pos_ego": (0.0, 0.0)
  }
}
```


(b)

Figure 1: An configuration of the *Parking* environment in the *HighwayEnv* simulator [14]. The bottom part (B) shows the configuration of the environment in JSON. On the top (A) is a rendered of that environment [3].

## 2 Background

### 2.1 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is an advanced subfield of Reinforcement Learning, a process that doesn't rely on labeled data for learning. Instead, the model is put in a simulation of the environment that it can interact with [27]. Depending on the actions the agent takes, it can be rewarded or punished, thus learning what the task is and how to perform it. In this paper, we focus on a specific scenario, namely perpendicular parking. Take a look at Figure 1a; this is the "game" that the agent will be playing. The goal is to park a car on a blue target while avoiding hitting the other cars. Figure 1B shows the JSON representation of the environment configuration, which includes parameters such as the number of lanes (`num_lanes`), the target parking spot index (`goal_lane_idx`), the ego vehicle's heading (`head_ego`), positions of other vehicles (`pvehicles`), and the starting position (`pos_ego`). Running these simulations is computationally expensive, each run takes around a minute depending on the hardware, so choosing the right environment is crucial.

As each simulation can result in a different amount of reward, scenarios in which the agent fails to achieve the task are the most valuable to us, since the agent can "learn" the most from them.

### 2.2 Surrogate Models

The aim of a surrogate model is to predict ahead of time what the outcome of running a DRL agent in the environment will be [3]. This allows us to classify the environments into "likely to fail" or "likely to pass" without running expensive simulations. A surrogate model is a simple, small model that is very cheap to run, allowing us to explore more environment configurations and select the ones that will result in the highest reward for the DRL agent. The surrogate model is trained on the DRL agent's outputs from initial training. In simple terms, we save each environment configuration that the DRL was trained on and assign it a label: fail or pass. This dataset is what the surrogate model is trained on.

### 2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep neural networks particularly effective for processing spatial data [12]. Unlike traditional Multi-Layer Perceptrons (MLPs) that treat input as flat vectors, CNNs preserve spatial relationships through convolutional operations.

CNNs offer several advantages for spatial problems:

- **Spatial feature extraction**: Convolutional layers can detect local patterns regardless of their position in the input [12].
- **Translation invariance**: Features can be recognized even when shifted to different locations [12].
- **Parameter efficiency**: Weight sharing in convolutional layers reduces the number of parameters compared to fully connected networks [12].

### 2.4 Genetic Algorithms for Test Generation

Genetic Algorithms (GAs) are population-based optimization techniques inspired by biological evolution [6]. In the context of DRL testing, GAs evolve a population of environment configurations to find those most likely to cause agent failures.

### 2.5 Attribution-Guided Mutation

The Indago framework employs saliency-based input attribution to guide mutations intelligently [3]. The saliency method computes gradients of the surrogate model's output with respect to each input parameter. These gradients indicate magnitude and direction. Magnitude represents influence of the parameter over the failure prediction. Direction shows whether increasing or decreasing a parameter value will increase failure likelihood

### 2.6 Related Work

The Indago framework, developed by Matteo Biagiola and Paolo Tonella, is a search-based approach designed for testing Deep Reinforcement Learning agents [3]. Its primary goal is to efficiently discover environment configurations that lead to failures of the DRL agent under test.

The process begins with collecting data, which consists of interaction data collected from a trained DRL agent interacting with its environment. This data consists of various environment configurations and the corresponding outcomes (e.g., success or failure of the task).

The collected interaction data is used to train a classifier, which acts as the surrogate model. This model learns to predict whether a given environment configuration is likely to cause the DRL agent to fail.

Next the surrogate model is used as a guide to a genetic algorithm that creates new, more challenging environments. Those environments are tested against the DRL agent to evaluate how difficult they are.

By using this surrogate-assisted approach, Indago aims to save significant computation time by deferring the execution of the full DRL agent simulation to only those configurations that are most promising for exposing failures. Experimental results reported by Biagiola and Tonella indicate that their search-based approach can find substantially more failures (e.g., 50% more) and more diverse failures compared to state-of-the-art techniques.

## 3 Methodology

This section describes our approach for developing and testing a CNN-based surrogate model for testing DRL agents. We build on Indago framework, introduced by Biagiola et al. [3]. Figure 2 shows a high-level overview of our approach. In the first stage we use DRL agent training history, where each data point is environment configuration, labeled as success (1) or failure (0). Next we transform originally tabular environment configuration, into spatial grid representations suitable for CNN processing. Afterwards, we design hybrid CNN that can be fed both image data and continuous features. We perform a grid search to find the best architecture, parameters and hyperparameter. Lastly we adapt the genetic algorithm to our CNN model and we perform an evaluation of our surrogate model.

### 3.1 Data Transformation

Figure 1a shows a visualization of the environment in which the DRL agent is trained, while Figure 1b shows the corresponding environment configuration used to generate this image. Each environment is defined by the following parameters:

- **num_lanes:** The size of the parking lot, indicating how many parking spots are available on each side. The maximum is 10 lanes.
- **goal_lane_idx:** The index of the target parking spot.
- **head_ego:** The initial angle of the ego vehicle, with a range of [0.0, 1.0).
- **pvehicles:** A list containing the indices of parking spots occupied by other vehicles.
- **pos_ego:** The initial x, y position of the ego vehicle, with a range of ((-10, 10), (-5,5)).

The training dataset consists of 10,000 environment configurations from the DRL agent's training history, each labeled '0' if the agent failed or '1' if it succeeded.

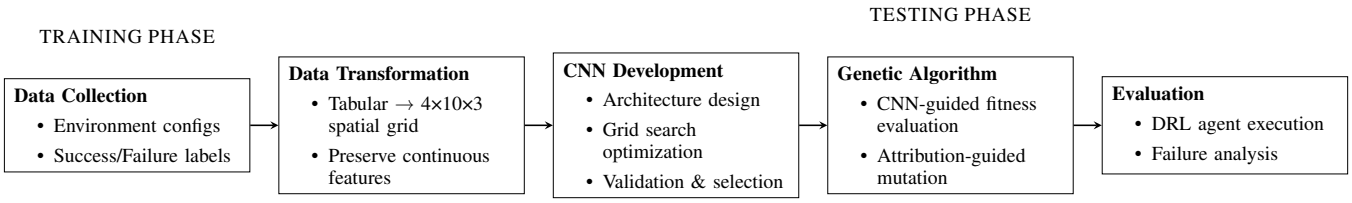



Figure 2: A high-level overview of our methodology, from data collection in the training phase to evaluation in the testing phase.

### 3.2 Data transformation

To use a Convolutional Neural Network (CNN), we first transformed the tabular data into an image format. We decided to use the most compact representation possible: a 4x10 grid with 3 channels. These channels represent the parked vehicles, the target spot, and an approximation of the vehicle's starting position. Image is an array of size 4x10x3, initialized with zeros. The first channel represents parked vehicles; for each occupied spot, we set the value at the corresponding grid index to 1. The second channel encodes the target parking spot in the same way. The third channel approximates the vehicle's starting position, with grid indices calculated based on the `num_lanes` parameter. We also appended the continuous features, `head_ego` and `pos_ego` to the grid. Those features will be concatenated to the convolutional layers output, ensuring no valuable information was lost.

### 3.3 CNN Design

We developed a custom CNN architecture to process our hybrid input, which consists of both the image representation and the continuous features. The grid part of the input is first passed through a series of convolutional layers. The output feature map is then concatenated with the continuous features from the input before being fed into a final dense layer. By design, our CNN is highly modular, CNN is created based on configuration dictionary that consist of:

- **conv_depth:** The number of layers in the convolutional block.
- **conv_width:** The number of filters in each convolutional layer.
- **dense_depth:** The number of layers in the final dense block.
- **activation_function:** The activation function to use ('relu', 'leaky_relu' or 'elu')[22] [18] [4].
- **leaky_slopes:** The negative slope for the Leaky ReLU activation.
- **drop_out_rate:** The dropout rate, applied uniformly across all layers [24].

A single 2D average pooling layer is placed between the final convolutional block and the first dense layer. We made a deliberate design choice to limit pooling to a single layer; given the compact 4x10 spatial dimensions of our input, further downsampling would risk the loss of critical spatial information necessary for accurate classification [12]. While our multi-stage grid search was extensive, it was necessarily bounded by computational constraints and did not cover the entire space of all possible CNN architectures. Therefore, the resulting model represents a well-performing architecture identified through a systematic search, rather than a globally optimal one.

### 3.4 CNN Training

Following the approach recommended by Biagiola and Tonella [3], we used data only from the second half of the DRL agent's training. With a test split of 0.2, this resulted in 4,000 training samples [7]. We used the training pipeline provided by the Indago framework, with one modification: we saved the model with the best validation loss instead of the best precision. Given our limited dataset size, we chose not to create a static validation set. Instead, promising models were tested directly in deployment against the DRL agent.

To identify the optimal architecture and hyperparameters, we performed a grid search [15]. We implemented a custom method, integrated with the Indago framework, to automate the generation of configuration files and a script to execute the grid search over both architectural parameters and training hyperparameters. The Indago framework identifies each unique model architecture with a single integer, referred to as a 'layer'. Manually defining configurations for the hundreds of architectures produced by our hyperparameter grid was infeasible. To automate this, we developed a deterministic mapping mechanism that treats the 'layer' integer as a linear index into the multi-dimensional hyperparameter space. The specific configuration for any given index is resolved programmatically using a series of integer division and modulo operations, effectively "unpacking" the index into a full architectural definition. Our script also performs a grid search over the learning rate, batch size, oversampling value, and weight decay.

### 3.5 Parameters of a grid search

We performed three sequential grid searches, each focusing on a different aspect of the CNN. The searches were run sequentially to narrow down the optimal configuration.

The first grid search was focused on the core architecture of the CNN:

- **Convolutional Depth and Width:** We tested a range of 1 to 4 convolutional layers with filter counts of 16, 24, 32, 48, and 64, This was informed by work from He et al. [12].
- **Dense Layer Depth and Width:** The dense layer was tested with 1 and 2 layers, with widths of 8, 16, and 32.

The second grid search focused on the activation function and regularization:

- **Activation Functions:** We tested three of the most popular activation functions: ReLU, Leaky ReLU, and ELU. For Leaky ReLU, we tested negative slope values of 0.05, 0.1, and 0.2 [22] [18] [4].
- **Regularization:** We tested three dropout rates (0.1, 0.3, and 0.5), as well as the inclusion or exclusion of batch normalization [9] [24].

The last grid search optimized training hyperparameters:

- **Learning Rate:** Tested values included 1e-4, 3e-4, 5e-4, and 1e-3 [10].
- **Batch Size:** We experimented with batch sizes of 128, 256, and 512 [20].
- **Weight Decay:** We tested the effect of L2 regularization by testing values of 0, 1e-5, 5e-5, and 1e-4 [17].

Parameters that were not the focus of a given grid search were set to a median value from the ranges of the subsequent searches. To pick the best architecture, we monitored standard metrics like precision, recall, F1 score, and Area under the ROC curve (AUROC), but gave significant weight to the analysis of the training and validation loss curves [5]. The most promising architectures were then evaluated in deployment, guiding the genetic algorithm and tested against the DRL agent.

### 3.6 Genetic Algorithm

The Genetic Algorithm (GA) implementation was provided by the Indago framework and uses a surrogate model to guide the selection and mutation of environments [3]. We adjusted its implementation to accommodate our new hybrid data representation. Originally its functions were built for a flat, tabular feature vector of 24 elements. Our transition to a hybrid grid-based representation expanded this vector to 124 elements, making the original mapping between attribution indices and environment parameters unusable.

The core of our adjustment was creating a new mapping mechanism. This logic translates an index from the CNN's 124-element flattened attribution vector back to its corresponding, structured environment parameter. For example:

- Indices 0-39 map `parked_vehicles_lane_indices`.
- Indices 40-79 map to the `goal_lane_idx`.
- Indices 80-119 map to the `position_ego`.
- Indices 120-123 map directly to the continuous features: `heading_ego`, `position_ego`, and `num_lanes`.

## 4 Implementation Details

Our methodology was implemented in Python 3.9, leveraging several open-source libraries. The core of our experimental pipeline was built upon the **Indago framework**, which provided the genetic algorithm and baseline MLP surrogate model implementation [3]. We developed our custom CNN surrogate model using **PyTorch** (version 1.8.0). Data manipulation and the spatial transformation logic were handled primarily with **NumPy** (version 1.21.0).

All experiments, including model training, grid searches, and the final evaluation runs, were conducted on a MacBook Pro with M1 chip and 16GB of RAM.

The complete codebase, including instructions for setting up the environment via a `requirements.txt` file, can be accessed at our public GitHub repository.

## 5 Study Design

To evaluate our CNN-based approach, we designed a two-phase study. The first phase focuses on answering RQ1 by identifying an effective CNN architecture and, determining the most reliable criteria for its selection. The second phase addresses RQ2 by conducting a direct comparative evaluation of our final CNN-guided search against the MLP-based baseline. This section includes the research questions, experimental procedure, and evaluation metrics for both phases of our research.

### 5.1 Research Questions

Our primary goal is to determine if a CNN, which is designed to process spatial information, offers an advantage over a standard MLP for surrogate-based DRL testing in an inherently spatial task like perpendicular parking. To achieve this, we define the following research questions:

- **RQ1:** What architectural and design choices lead to effective CNN-based surrogate models for DRL testing?
- **RQ2:** How does the performance of the CNN-based surrogate model compare to an MLP-based surrogate baseline?

### 5.2 Case Study and Experimental Setup

Our study is centered on the environment from the `highway-en` simulator, a perpendicular parking task previously described in Section 3.1 [14]. The DRL agent was trained using Hindsight Experience Replay (HER) [3]. It's objective is to park a vehicle in a designated spot while avoiding collisions.

#### Models for Comparison

We compare the performance of two distinct approaches for generating failure scenarios:

1. **CNN-based Search (Our Approach):** The genetic algorithm described in Section 3.3, guided by our custom CNN surrogate model. This approach uses the 4x10x3 grid-based data representation transformed from the original environment parameters.
2. **MLP-based Search (Baseline):** The same genetic algorithm guided by an MLP surrogate, replicating the core method of Biagiola and Tonella [3]. The MLP is trained on the original, non-transformed tabular data.

### 5.3 Procedure

Our procedure is divided into two phases, one for each research question.

**Phase 1: CNN Model Development and Selection (for RQ1).**
The aim of Phase 1 was to find the most effective CNN architecture. We performed a 3-stage grid search. The first and widest grid search was designed to find the most suitable CNN architecture. To achieve that, we selected the 8 best-performing architectures based on standard validation metrics (Precision, Recall, F1-score, AUROC). We tested these architectures in deployment and found no correlation between their validation metrics and their actual failure discovery rate. This finding led us to form a new hypothesis: a model's generalization capability, indicated by a smaller gap between training and validation loss curves, is a more reliable predictor of its effectiveness. Based on this hypothesis, we selected 5 more architectures with promising learning curves and tested their effectiveness. As this method proved superior, we selected our final CNN model for Phase 2 by prioritizing strong generalization over raw metric scores

**Phase 2: Comparative Evaluation (for RQ2).**
In this phase, we used the final CNN model selected in Phase 1 to conduct a comparison against the baseline. For each of the models, we performed multiple independent runs to account for the stochastic nature of the search algorithms. In each run, the designated approach was tasked with generating 50 potentially failure-inducing environments. The search for each environment was limited by a fixed time budget. Each environment proposed by the search was then executed with the actual DRL agent in the simulator to verify whether it resulted in a true failure (e.g., a collision or timeout).

### 5.4 Evaluation Metrics

To quantitatively answer our research questions, we used different sets of metrics for each phase of the study.

**Metrics for RQ1: Model Selection**
During the model development and selection phase, the following criteria were used to identify the final CNN architecture:

- **Classification Metrics:** Standard metrics including Precision, Recall, F1-score, and Area Under the ROC Curve (AUROC) were used for the initial selection of candidate models.
- **Learning Curve Analysis:** The primary criterion for final model selection was an analysis of the model's learning curves, specifically the gap between training loss and validation loss, which served as a proxy for generalization.
- **Deployment Failure Rate:** The final measure of a candidate model's effectiveness was its success rate in guiding the GA to find failing test cases during preliminary deployment tests.

**Metrics for RQ2: Comparative Performance**
To compare the final selected CNN against the MLP baseline, we employed the comprehensive set of metrics established by Biagiola and Tonella [3]:

- **Mean Number of Discovered Failures:** For each independent run, we count the total number of environments (out of the 50 generated) that result in a verified failure. The final reported metric is the mean of these failure counts taken across all runs. This directly measures the effectiveness of each approach in finding failures and serves as the primary point of comparison for RQ2.
- **Input Diversity:** To assess how varied the generated failure *configurations* are, we clustered their vector representations. We use the same methodology applied by Biagiola and Tonella [3], we used k-means clustering [16] with silhouette analysis [23] to determine the optimal number of clusters. We then reported:
  - *Coverage:* The percentage of clusters that contain at least one failure generated by the given approach.
  - *Entropy:* A measure of how uniformly the failures are distributed across the covered clusters.
- **Output Diversity:** To measure the diversity of the DRL agent's *behavior* in response to the failures, we performed the same clustering analysis on the agent's failure trajectories. We again reported *Coverage* and *Entropy* for the resulting trajectory clusters. These diversity metrics show how diverse agent behaviors are. This is essential for answering RQ2.
- **Statistical Analysis:** To ensure our conclusions are robust, we used the Mann-Whitney U Test [19] to determine if the observed differences in failure counts and diversity metrics between the approaches are statistically significant ($p < 0.05$). To complement the significance test, we also measured the effect size using the Vargha-Delaney A statistic [25].

## 6 Results

### 6.1 RQ1: Architectural and Design Choices for Effective Surrogates

**CNN Architecture Search**
The first grid search resulted in 160 architectures tested. We evaluated 8 of them in deployment by generating 50 parking environment configurations and testing them against the DRL agent. For each of these, we saved the probability of generating a failing environment. We selected these architectures by taking the 2 best performing models for each metric: precision, recall, F1, and AUROC.

The results are summarized in Table 1. As we can observe, the validation metrics do not translate to deployment performance. For instance, architecture 5 has the best F1-score (0.29) but a low failure rate of 0.21. On the other hand, architecture 4 has a poor F1-score of 0.17 but achieved the highest failure rate of 0.62, making it the best model we tested. This suggests that for this problem, optimizing for traditional classification metrics is not a reliable strategy for model selection.

This led us to hypothesize that a model's generalization capability is a more important for good performance. This hypothesis is supported by an analysis of the learning curves. Figure 3 shows the learning curve for the best-performing model (Architecture 4), where the small gap between the training and validation loss indicates strong generalization.

In contrast, Figure 4 shows the curve for a poorly performing model (Architecture 8), where a significant divergence between the curves indicates overfitting.

To test this new hypothesis, we conducted a small exploratory study. We selected five additional models based solely on their promising learning curves (i.e., a minimal generalization gap). These models achieved an average failure discovery rate of 0.48. This represents a 65% improvement over the 0.29 average rate of the models originally selected with validation metrics, confirming that learning curve analysis is a more reliable selection criterion.

Table 1: Results from the initial architecture search. Models were selected based on top validation metrics (P: Precision, R: Recall, F1: F1-Score, AU: AUROC). The final columns show validation loss (L) and deployment failure rate (F). Note the lack of correlation between high validation scores and deployment performance (e.g., ID 4 vs. ID 5).

| ID | Conv Depth | Conv Width | Dense Depth | Dense Width | P | R | F1 | AUC | L | F |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 32 | 2 | 32 | 0.32 | 0.23 | 0.27 | 0.71 | 0.49 | 0.27 |
| 2 | 4 | 48 | 2 | 32 | **0.34** | 0.19 | 0.25 | 0.69 | 0.53 | 0.43 |
| 3 | 2 | 64 | 1 | 32 | 0.10 | **0.61** | 0.18 | 0.71 | 0.46 | 0.17 |
| 4 | 1 | 32 | 2 | 32 | 0.10 | **0.61** | 0.17 | 0.70 | 0.44 | **0.62** |
| 5 | 4 | 64 | 2 | 48 | 0.25 | 0.36 | **0.29** | 0.71 | 0.45 | 0.21 |
| 6 | 3 | 48 | 1 | 48 | 0.25 | 0.34 | **0.29** | 0.70 | 0.49 | 0.27 |
| 7 | 2 | 24 | 1 | 16 | 0.10 | 0.59 | 0.17 | **0.71** | **0.41** | 0.37 |
| 8 | 3 | 64 | 1 | 48 | 0.15 | 0.45 | 0.23 | **0.71** | 0.46 | 0.43 |

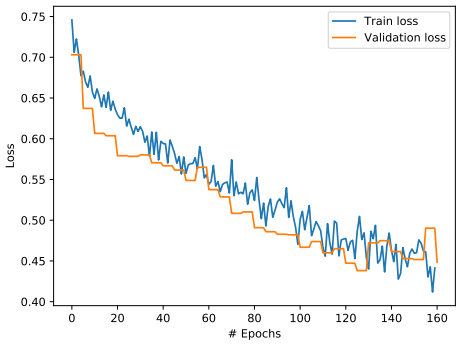



Figure 3: Learning curve for Architecture 4 (Failure rate = 0.62). The small gap between training and validation loss shows good generalization and led to the best performance.

### Final Architecture and Design Choice

After choosing the best architecture through learning curve analysis, we conducted two more grid searches for activation functions and training hyperparameters. We continued to prioritize models that exhibited strong generalization while also monitoring validation metrics like AUROC to select the final configuration.

The final configuration consists of: 1 convolutional layer with a width of 32, 2 dense layers with a width of 32, Leaky ReLU as the activation function with a slope of 0.05, a dropout rate of 0.1, with batch normalization active, a batch size of 256, learning rate of 0.001 and a weight decay of 0.00001.

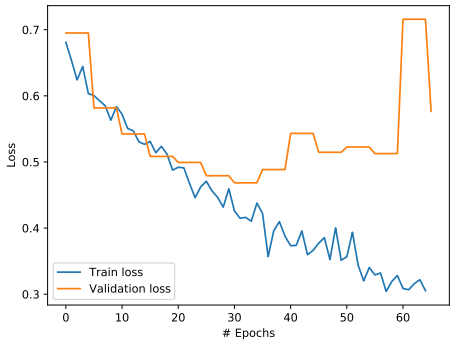



Figure 4: Learning curve for Architecture 8 (Failure rate = 0.43). The large gap shows the model is overfitting, which resulted in worse performance.

This final model achieved a validation loss of 0.48, a precision of 0.12, a recall of 0.51, an F1-score of 0.20, and an AUROC of 0.71.

Therefore, the answer to RQ1 is twofold. The most effective architecture was a shallow CNN (1 convolutional layer, 2 dense layers, with the parameters listed above). However, the most critical design choice was the selection methodology: prioritizing models with strong generalization, guided by their learning curves, is more effective than selecting models based on traditional classification metrics.

## 6.2 RQ2: Comparison with MLP Baseline

To answer RQ2, we conducted a direct comparison between our final CNN-guided Genetic Algorithm (GA) and the MLP-guided GA baseline, as described in our Study Design. Both approaches were run for 50 independent trials, and their ability to find failure-inducing environments was measured in terms of failure count and diversity. This allows us to quantify the benefits of our spatial data representation and CNN architecture over a standard tabular approach.

### Number of Discovered Failures

First, we compared the total number of unique failing environments discovered by each approach. As shown in Table 2, our CNN-guided search found a mean of **25.76** failures per run, a statistically significant improvement over the 14.98 failures found by the MLP baseline. The large effect size confirms that this improvement is not due to chance, but is a direct result of the more effective guidance provided by the CNN surrogate.

### Diversity of Discovered Failures

Beyond the raw count, we evaluated the diversity of the discovered failures in both the input space (environment configurations) and the output space (agent behaviors).

**Input Diversity.** The results for input diversity, shown in Table 2, demonstrate the most significant advantage of our approach. The CNN-guided search achieved **70.0%** coverage of the input failure clusters and an entropy of **19.6%**. This

Table 2: Comparison of CNN vs. MLP surrogate performance over 50 runs. All differences are statistically significant (Mann-Whitney U test, $p < 0.001$) with a large effect size (Vargha-Delaney A).

| Metric | Sub-Metric | MLP (Baseline) | CNN (Ours) |
|---|---|---|---|
| Failure Rate | Mean Failures Discovered | 14.98 | **25.76** |
| Input Diversity | Coverage (%) | 59.0% | **70.0%** |
| | Entropy (%) | 6.8% | **19.6%** |
| Output Diversity | Coverage (%) | 47.6% | **68.7%** |
| | Entropy (%) | 58.2% | **69.4%** |

is a significant improvement over the MLP baseline, which only covered 59.0% of clusters with a lower entropy of 6.8%. This finding strongly supports our hypothesis that a CNN can more effectively explore the spatial dimensions of the problem, leading to a much more comprehensive and varied set of test cases.

The comparison of output (behavioral) diversity, presented in Table 2, also shows a clear and statistically significant advantage for the CNN model. It achieved higher coverage (68.7% vs. 47.6%) and higher entropy (69.4% vs. 58.2%) than the MLP baseline. This indicates that the more diverse inputs found by the CNN also successfully trigger a wider and more varied set of failing behaviors from the DRL agent.

In conclusion, our results for RQ2 show that the CNN-based surrogate significantly outperforms the MLP baseline in the most critical aspects of test case generation: finding a greater number of failures and ensuring those failures are drawn from a much more diverse set of input configurations. This validates our core hypothesis that transforming the problem into a spatial domain and using a CNN is a more effective strategy for testing this DRL agent.

# 7 Threats to Validity

## 7.1 Internal Validity

Both models had the same allocated budget, but environment generation as well as the testing against DRL agents were performed on different machines. This can result in subtle differences in performance due to difference in hardware. To mitigate this, both models were trained on the same data. For each model, the same number of environments were generated and tested against the same DRL agent.

## 7.2 External Validity

Our results are based on only one DRL environment. This environment is widely used in DRL community as a benchmark, but generalization to other environments is not guaranteed. Future work should focus on other benchmark environments like DonkeyCar simulator or environments from Mujoco simulator [3].

## 7.3 Conclusion Validity

To ensure the reliability of our findings and account for the stochastic nature of the genetic algorithm, we performed 50 independent runs for both our approach and the baseline. This follows established guidelines for the empirical assessment of randomized algorithms [2]. Furthermore, our conclusions are supported by sound statistical analysis, including the Mann-Whitney U test and the Vargha-Delaney effect size statistic, as recommended in the literature[25] [19].

A remaining threat to conclusion validity stems from the hyperparameter optimization process. Due to computational constraints, the grid search for the optimal CNN architecture and hyperparameters was not exhaustive. While the search process itself is deterministic, the training of each candidate model involves stochastic elements (e.g., weight initialization). The primary limitation is that the explored hyperparameter space was restricted. Consequently, the final CNN architecture may be sub-optimal, which could lead to an underestimation of the true effect size of our proposed approach compared to a fully optimized model

# 8 Conclusion and Future Work

Testing Deep Reinforcement Learning (DRL) agents in safety-critical systems like autonomous driving is essential, but the high computational cost of simulation makes finding challenging test cases difficult. While prior work has demonstrated the value of using surrogate models to guide this search, standard MLP-based approaches cannot fully leverage the spatial nature of environments like a parking lot.

In this thesis, we addressed this gap by introducing two primary contributions. First, we developed a novel testing approach that transforms tabular environment data into a 2D grid, enabling the use of a Convolutional Neural Network (CNN) as a more effective surrogate. Second, we identified a more reliable design choice for model selection: our grid search showed that a model's generalization capability, as indicated by its learning curves, is a far better predictor of deployment performance than standard validation metrics like F1-score or AUROC.

Our experiments provided clear answers to our research questions. For RQ1, we demonstrated that a shallow CNN architecture, selected based on its strong generalization properties, is highly effective for guiding the genetic algorithm. For RQ2, the comparison against the baseline confirmed the superiority of our approach. The CNN-guided search discovered a mean of 25.76 failures per run, a 72% improvement over the 14.98 failures found by the MLP. This advantage extended to diversity across all metrics; for example, input diversity entropy almost tripled from 6.8% to 19.6%.

These findings validate our central hypothesis: for problems with inherent spatial characteristics, representing the data spatially and using a CNN is a more effective strategy for testing DRL agents. While our study was limited to a single parking environment, it lays a strong foundation for future work. Promising directions include applying this technique to other spatial DRL domains, such as robotics [8] or drone navigation [26] , and incorporating diversity metrics directly into the fitness function to further enhance the comprehensiveness of the test case generation[3].

# 9 Responsible Research

## 9.1 Ethical Considerations

The primary ethical motivation of this research is to improve the safety and reliability of DRL agents in safety-critical applications, such as autonomous driving. By developing a more efficient method for discovering failures, this work aims to contribute to more robust and dependable autonomous systems.

A key concern is the potential for algorithmic bias. The performance of the surrogate models is fundamentally dependent on the training data generated from the DRL agent's interactions. If this dataset lacks diversity, the surrogate model may learn to effectively predict only a subset of possible failures, which could create a false sense of security. While our results show the CNN approach enhances the diversity of discovered failures compared to the MLP, the risk of undiscovered failures persists.

The transparency of deep learning models like our CNN is an important consideration. The "black box" nature of these models can make their predictions difficult to fully interpret, which poses a challenge when they are used to validate safety-critical systems.

To mitigate these concerns, we have committed to responsible research practices. As mentioned in the "Reproducibility" section, our full codebase, data, and models are publicly available. This transparency enables independent scrutiny and validation by the broader research community, which is a critical safeguard for ensuring the responsible development of AI technologies.

## 9.2 Reproducibility

We have published all of our code, data, and grid search results in our public GitHub repository[1]. We used publicly available packages and libraries, and we have extensive documentation in the repository about the setup and running of the project.

## 9.3 Use of LLMs

LLMs were used to improve flow of text and grammatical mistakes only. They didn't have any contribution to the contents of this research paper.

# References


[1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety, 2016.

[2] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, page 1–10, New York, NY, USA, 2011. Association for Computing Machinery.

[3] Matteo Biagiola and Paolo Tonella. Testing of deep reinforcement learning agents with surrogate models. *ACM Transactions on Software Engineering and Methodology*, 2024.

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[5] Sarah Farhadpour, Timothy A. Warner, and Aaron E. Maxwell. Selecting and interpreting multiclass loss and accuracy assessment metrics for classifications with class imbalance: Guidance and best practices. *Remote Sensing*, 16(3):533, 2024.

[6] Davide Farinati and Leonardo Vanneschi. A survey on dynamic populations in bio-inspired algorithms. *Genetic Programming and Evolvable Machines*, 25(2):19, 2024.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[8] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, 2016.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[11] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey, 2021.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Yu Lei, Zhitao Wang, Wenjie Li, Hongbin Pei, and Quanyu Dai. Social attentive deep q-networks for recommender systems. *IEEE Transactions on Knowledge and Data Engineering*, 34(5):2443–2457, 2022.

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

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

[16] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[18] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *Proceedings of ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[19] Henry B. Mann and Donald 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.


[1] https://github.com/Braszczynski/CNN-surrogate-reloaded

[20] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[22] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

[23] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[25] Andreas Vargha and Harold D. Delaney. A critique and improvement of the CL common language effect size statistics. *Journal of Educational and Behavioral Statistics*, 25(1):101–132, 2000.

[26] Junqiao Wang, Zhongliang Yu, Dong Zhou, Jiaqi Shi, and Runran Deng. Vision-based deep reinforcement learning of uav autonomous navigation using privileged information, 2024.

[27] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.