Surrogate Reloaded: LSTM-Based Failure Prediction for Testing DRL Agents

Steven van den Wildenberg[1]
Supervisor(s): Dr. A. Panichella[1]**, A. 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: Steven van den Wildenberg
Final project course: CSE3000 Research Project
Thesis committee: Dr. A. Panichella, A. Bartlett, Dr. P. Kellnhofer

Surrogate Reloaded: LSTM-Based Failure Prediction for Testing DRL Agents

Abstract

Testing Deep Reinforcement Learning agents for safety and performance failures is critical but computationally expensive, requiring efficient methods to discover failure-inducing scenarios. Indago, a state-of-the-art testing framework, addresses this by using a Multi-Layer Perceptron (MLP) surrogate model to guide a search algorithm towards potential failures. However, the MLP considers only static initial environment configurations. We propose replacing the static MLP with a Long Short-Term Memory (LSTM) network to leverage the temporal data from agent interaction sequences during training. We evaluated our LSTM-based surrogate against the original MLP within the Indago framework on the HighwayEnv autonomous driving benchmark. Although the best LSTM exhibited lower predictive precision on a held-out test set (11.5% vs. 24%), its integration into the search-based testing pipeline led to a higher failure rate and input diversity. The LSTM guided search discovered more failures on average (34.6% vs. 30%) and explored the input space more thoroughly, achieving 99% cluster coverage and a tenfold increase in input entropy compared to the MLP. Interestingly, training on very short sequences of only two timesteps produced the most effective models, a finding that challenges the initial hypothesis that longer temporal contexts are beneficial for improved performance.

1 Introduction

Deep Reinforcement Learning (DRL) agents are often trained through trial-and-error within simulated environments, combining the power of deep neural networks with the reward-driven optimization of Reinforcement Learning (RL). While powerful, DRL agents can fail by violating environmental constraints, breaching safety rules, or failing to meet performance thresholds [2, 13]. The need to test these systems is critical, particularly in high-stakes domains like safety-critical robotics, autonomous driving, and financial trading, where failures can lead to significant real-world consequences [11, 21, 22]. However, testing DRL agents is challenging. It is computationally expensive and requires generating a vast number of diverse scenarios to uncover potential weaknesses.

To address the inefficiency of testing, researchers have moved from random or sampling-based environment generation towards more intelligent search methods [25]. A recent approach is the state-of-the-art *Indago* framework by Biagiola et al., which uses a surrogate model to predict likely failure-inducing environments [4]. The main benefit of this approach is that the surrogate can estimate the likelihood of failure for a test scenario without running the full simulation, which would otherwise take seconds to minutes per run and make large-scale testing excessively slow. After training a Multi-Layer Perceptron (MLP) on existing training data, Indago employs search algorithms like Hill Climbing or a Genetic Algorithm to efficiently discover new and diverse failing test cases. This method proved significantly more effective than prior techniques, finding more failures that elicited more varied agent behaviors. While Indago marks a significant step forward, its exploration was limited to using an MLP as the surrogate. The training logs of DRL agents, however, contain not just static initial environments but also timeseries data detailing the agent's interactions over time. This sequential data remains an unexplored resource. This paper investigates whether a surrogate model designed for sequential data, the Long Short-Term Memory (LSTM) network, can leverage this temporal information to improve the failure discovery process. LSTMs are

a type of Recurrent Neural Network (RNN) that excels at capturing temporal dependencies [14], making them a natural candidate for modeling DRL agent interactions.
We consider the following research question:

> **RQ1:** *How effectively can LSTM surrogate models predict failure configurations for testing DRL agents compared to MLP surrogates?*

This is further refined into two sub-questions:

- **RQ1.1:** *What is the internal predictive precision of the LSTM surrogate model for identifying failure-inducing environment configurations?*
- **RQ1.2:** *How does the use of the LSTM surrogate model in the Indago testing framework impact the number and diversity of discovered failure scenarios compared to the baseline MLP surrogate?*

We conduct our experiments in a simulated autonomous parking environment called *HighWay-Env* [16]. The scenario involves a vehicle navigating into a designated parking spot within a parking lot of 20 spaces, some of which are randomly occupied. Our results show that, although the LSTM does not surpass the MLP in predictive precision, it proves effective within the Indago framework. The best performing LSTM model discovered more failures than the MLP baseline (34.6% vs. 30% on average), while exploring the input space more broadly, achieving 99% cluster coverage and a tenfold increase in input entropy. Interestingly, this was achieved with models trained on short sequences (only two timesteps), while longer sequences were notably absent. Our key contributions are: 1) The integration of an LSTM-based surrogate into the Indago testing pipeline, 2) An empirical comparison with the MLP baseline, revealing strengths and limitations of sequential modeling in this setting.

This paper is structured as follows. Section 2 provides related work and background information on different concepts used throughout the paper. Section 3 details our specific contributions. Section 4 outlines the study design, including our dataset and evaluation protocol. Section 5 presents the empirical results for our research questions. Section 6 provides possible threats to validity. Section 7 offers conclusions and directions for future work, and, finally, section 8 discusses responsible research considerations.

2 Background and Related Work

2.1 Reinforcement Learning

Reinforcement learning (RL) represents a class of algorithms where an autonomous agent learns the ability to make sequential decisions. It does this by interacting with an environment, aiming to get the most rewards over time [24]. Unlike supervised learning, which uses pre-labeled data, or unsupervised learning, which looks for hidden patterns, RL learns through trial and error. This learning process is typically set up as a Markov Decision Process (MDP). An MDP has four key components: states, actions, transition probabilities, and a reward function. The agent's main goal is to figure out the best strategy, or policy ($\pi*$), that will maximize the total discounted rewards it expects to receive. This approach captures real-world situations where the best choices depend on the current circumstances and have long-term effects. This sequence of states, actions and rewards is called an *episode*. An episode terminates successfully when the objective is reached, and unsuccessfully when an exit criterion, for example constraint violation, a time-step limit, or a negative cumulative reward bound, is met.

2.2 Deep Reinforcement Learning

Deep reinforcement learning (DRL) combines reinforcement learning with deep neural networks to handle high-dimensional state and action spaces that traditional RL methods cannot effectively process [20]. Where classical RL algorithms like Q-learning use tabular representations that become intractable for large state spaces, DRL uses neural networks as function approximators to estimate value functions or policies directly from input. Neural networks enable RL agents to learn complex representations from high-dimensional inputs such as images or continuous sensor data.

2.3 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks are a specialized type of Recurrent Neural Network (RNN) designed to effectively model temporal sequences and learn long-range dependencies [14]. Standard RNNs, while capable of processing sequential data by maintaining an internal state or "memory", often struggle to use information that lies far back in the sequence. This difficulty arises primarily from the *vanishing gradient problem*, where the influence of past inputs diminishes too rapidly during the network's training process, making it hard to learn connections over extended time periods.

LSTMs were specifically engineered to overcome this limitation. At their core, LSTMs introduce a more sophisticated memory mechanism within each recurrent unit. This includes a cell state, allowing information to flow through the network relatively unchanged over many timesteps. LSTMs use three gating mechanisms: an input gate, a forget gate, and an output gate. These gates are small, learnable neural networks that regulate the flow of information into and out of the cell state: the *input gate* controls what information enters the cell state, the *forget gate* controls what is discarded from the cell state, the *output gate* controls what part of the cell state is added to the current timestep output. This gated architecture enables LSTMs to selectively remember important information for long durations and discard irrelevant data. This allows them to capture dependencies across hundreds of time steps. Consequently, LSTMs are well-suited for tasks involving time-series data, such as predicting events based on subtle patterns spread out over time. Their ability to handle variable-length sequences further enhances their use in diverse temporal modeling scenarios.

2.4 Genetic Algorithms

A Genetic Algorithm (GA) maintains a *population* of candidate solutions (*chromosomes*) whose quality is measured by a *fitness function* [10]. Each generation selects the fittest individuals, recombines their genes (crossover), and applies random *mutation* before evaluating the offspring and forming the next population. Iterating this cycle explores the search space without gradients and typically converges toward high fitness regions. In our experiment, a chromosome is a parking lot environment configuration as described in Section 3.1. In our setting, crossover mixes elements such as car positions or goals from two configurations, while mutation tweaks individual features, like vehicle orientation or location, to produce potential failure configurations.

2.5 Related Work

Prior work on testing DRL agents through environments use vanilla Monte-Carlo method, which struggles to uncover failures. Uesato et al. [25] use a sampling approach using a trained classifier with failure probability estimation. Another work deploys search to generate mazes based on the performance of the DRL agent that navigates through the

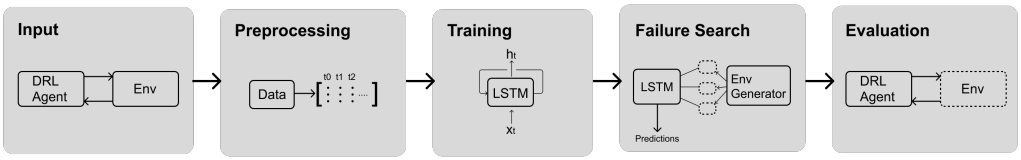


Figure 1: Overview of our testing pipeline for DRL agents using an LSTM surrogate. We transform interaction data from DRL agents into input sequences for LSTM training. Once trained, the LSTM predicts failure likelihoods and guides a Genetic Algorithm to generate high-risk environment configurations, which we use to evaluate the DRL agent.

mazes [9]. Although this requires the DRL agent to perform on the generated environment, which can be computationally expensive.

STARLA evolves entire trajectories with a multi-objective GA targeting low reward, high fault probability and action uncertainty, uncovering more faults than random testing. Although only for deterministic DQN agents with discrete actions [27].

The Indago approach by Biagiola et al. [4] trains a cheap surrogate on the agent's own traces to guide search over environment configurations. Using the surrogate's failure probability as a fitness signal, they find 50% more distinct failures than random or adversarial sampling. The surrogate improves DRL testing by predicting scenario outcomes, greatly reducing the number of expensive DRL agent executions, which could otherwise take seconds to minutes per run. The significant speedup of the surrogate motivates further exploration of its landscape. Yet, their use of a basic MLP on static features leaves temporal dynamics unexplored. We integrate with Indago to address this knowledge gap.

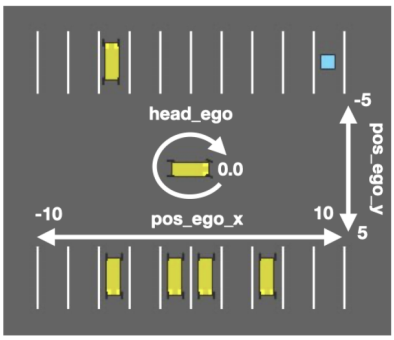
LSTMs in Related Applications. The use of LSTMs for modeling temporal phenomena is well established, though existing applications differ from our approach in key aspects. In failure detection, for instance, LSTMs serve as passive monitors to predict faults from sensor streams, rather than as active tools to generate failure-inducing inputs [5, 7]. Similarly, when used as surrogate models to replace costly simulators, they typically model entire system trajectories and are not integrated into an adversarial search loop based on initial conditions [12, 6]. Finally, while LSTMs are effective for vehicle trajectory prediction, they act as forward predictors of behavior, not as a surrogate objective to guide systematic test-case generation from a single initial state [1, 18].

3 Methodology

This section outlines the experimental pipeline used in this study. We build on the Indago framework of Biagiola et al. [4], adopting its GA search. A full overview of the approach can be found in Figure 1. Starting from the training data collected from DRL agent interactions, sequences are extracted and used to train the LSTM surrogate model. The search of a Genetic Algorithm utilizes this model, to identify environment configurations predicted to induce agent failures. We validate these configurations by running the DRL agent on them in the actual environment.

3.1 Parking Environment Use Case

We focus on the *Parking* scenario from *HighwayEnv* [16], in which a DRL agent controls an *ego-vehicle* inside a rectangular parking lot (see Figure 2). At the beginning of each episode,



A

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

B

Figure 2: A visualization of a the Parking environment with its corresponding configuration (reproduced from Biagiola et al. in the *Parking* environment from *HighwayEnv* [4][16]). The left-hand side (A) shows how the configuration on the right-hand side (B) is rendered in the environment.

the vehicle starts at position `pos_ego`, which defines the starting coordinates. The heading `head_ego`, is the rotation of the vehicle and lies within $[0.0, 1.0)$, where 0.0 is perpendicular to the parking spaces. The agent must drive to a designated parking spot `goal_lane`, avoiding parked cars specified by `pvehicles`, which indicate which of the 20 parking spots are occupied. The agent acts through continuous *throttle* and *steering* commands and receives a negative reward proportional to the Euclidean distance to the target, a positive final reward for a successful park, or a large penalty for collisions or exceeding the time limit. An episode therefore ends through *success*, *collision*, or *timeout*. These *collision* and *timeout* outcomes form the failure label, while the *success* represents itself.

3.2 Data Collection and Filtering

The episodic logs of the DRL agent consist of the environment configuration, the throttle and steering actions of the agent, the reward received, the cumulative speed, and the car trajectory coordinates. Due to the inherent exploration behavior of DRL agents, early episodes often contain stochastic actions. This can cause failures unrepresentative of the agent's learned behavior. We address this using the training progress filter from the Indago framework. This filters out a percentage of earlier episodes from the data.

3.3 Feature engineering and augmentation

The progress of the agent's behavior leads to a class imbalance. Such a class imbalance, where there are far fewer failures than successes, causes overfitting on the majority class. To tackle this, we use weighted loss functions [15] and random oversampling [3], which are part of the Indago framework. Weighted loss functions give larger influence to the minority class, the failures. Oversampling duplicates minority class samples, with a parameter controlling how many are added relative to the majority class.

To feed the interaction data as input to the LSTM, we transform it into a feature vector with multiple timesteps. The initial timestep follows the same transformation as the MLP surrogate baseline. The vector features in order: the `goal_lane` and `head_ego`, a one-hot encoding of the `pvehicles`, and finally the two coordinates of the `pos_ego`. The one hot encoding is a sequence of 20 values with 1 being occupied and 0 being unoccupied.

To evolve the configuration over time, we only change the `pos_ego`, which updates using the car trajectory coordinates that we get from the dataset. We approximate the heading of the vehicle at each timestep using the instantaneous direction of motion, including normalization:

$$\theta(t) = \frac{(\text{atan2}(\Delta y, \Delta x) + 2\pi)\%2\pi}{2\pi} \tag{1}$$

This assumes the heading follows the trajectory direction, which is reasonable in typical driving scenarios with small slip angles. The *atan*2 function's interval is $[-\pi, \pi]$. We normalize this to the interval of [0, 1) which is used by the initial environment configuration `heading_ego`.

We experimented with alternative representations, such as concatenating the static configuration with timeseries data (action, speed, trajectory). However, we discarded these approaches due to their poor predictive performance (validation AUC ≤ 0.5). In addition, the evolving initial configuration approach simplifies inference because it avoids masking missing data during evaluation and GA search.

Final Input Representation The final input to the LSTM is a timeseries encoding a single episode. Each timestep consists of the same feature representation: `goal_lane` (scalar), `head_ego` (scalar), a one-hot vector of `pvehicles` (20-dimensional), and the two positional coordinates of `pos_ego` (2-dimensional). This results in a feature vector of dimension 24 per timestep.

We generate the full sequence by evolving only `pos_ego` and `head_ego` according to the car trajectory in the DRL logs, while keeping `goal_lane` and `pvehicles` constant across all timesteps. This produces a tensor of shape $T \times 24$, where T is the number of recorded timesteps for the episode. At inference time, we use only the first timestep ($T = 1$), simulating a static initial configuration, to predict failure likelihood.

3.4 LSTM Surrogate model

The surrogate is implemented as a unidirectional LSTM network followed by mask-aware global *mean* pooling and a shallow feed-forward classifier [8]. We use mean pooling because it creates a more stable and representative summary of the entire sequence, especially for short or varying sequences.

Architecture Let the input be a variable-length sequence $\mathbf{x}_{1:L} \in \mathbb{R}^{L \times d}$, where d is the feature dimension and $L \leq L_{\max}$. The network computes hidden states $\mathbf{h}_{1:L} \in \mathbb{R}^{L \times h}$ with an n-layer LSTM:

$$\mathbf{h}_{1:L} = \text{LSTM}_{n,h}(\mathbf{x}_{1:L}).$$

A mask-aware mean operation produces a fixed-size representation

$$\bar{\mathbf{h}} \;=\; \frac{1}{L} \sum_{t=1}^{L} \mathbf{h}_t.$$

The pooled vector passes through a fully connected layer of size f with ReLU and dropout p, followed by a linear layer that outputs two logits $\mathbf{z} \in \mathbb{R}^2$ for the classes FAILURE / SUCCESS.

Training objective and optimization We minimize a weighted softmax cross-entropy loss, optimize parameters with Adam (learning-rate ℓ), and training stops early if the precision does not improve for k consecutive epochs (restoring the best weights).

3.5 Hyperparameter Grid Search

In line with multi-fidelity optimization work [17], we employ a three stage setup that progressively increases the number of random seeds while narrowing the set of candidate hyperparameter configurations:

- Stage 1: Every configuration is trained on one seed and ranked by validation AUC
- Stage 2: The top 150 configurations are retrained with two additional seeds (three seeds in total). The ranking is updated using the mean AUC over these three runs.
- Stage 3: The 15 best configurations from Stage 2 are evaluated with seven further seeds, giving mean ± s.d. AUC over ten seeds.

This stratified strategy of random seeds balances statistical reliability and computational feasibility. Evaluating every hyperparameter configuration on ten seeds would increase the grid search cost by a significant amount. Therefore we first screen on a single seed and increase the seeds for the most promising configurations.

We use the *area under the ROC curve* (AUC), the probability that a randomly chosen positive instance scores higher than a negative one as the selection metric for an immediate validation on whether a model is doing better than chance [23]. For context, *precision* is the share of predicted positives that are true, *recall* is the share of actual positives recovered, and the *F1* score is their harmonic mean [23]. AUC has a well-defined lower bound: a score of 0.5 corresponds to a random classifier, while values above 0.5 indicate discriminatory power between positive and negative classes. When performance is near chance, threshold dependent measures such as precision, recall, or F1 can be misleading or uninformative.

3.6 Surrogate Selection

In order to evaluate the surrogate model we will use a held-out test set. Before the grid search we take a percentage of the dataset. The test set also has a progress filter, which excludes a percentage of the episodes, because earlier failures do not always represent later performance well. This filter does not necessarily have to be the same as for the training data set. A smaller filter would likely decrease the performance of the model on the test set. However, this is not necessarily a problem, because precision/recall can drop, yet the relative ordering of environments, which is important for search, remains informative.

4 Study Design

We consider the following research question:

> **RQ1:** *How effectively can LSTM surrogate models predict failure configurations for testing DRL agents compared to MLP surrogates?*

This is further refined into two sub-questions:

- **RQ1.1:** *What is the internal predictive precision of the LSTM surrogate model for identifying failure-inducing environment configurations?*
- **RQ1.2:** *How does the use of the LSTM surrogate model in the Indago testing framework impact the number and diversity of discovered failure scenarios compared to the baseline MLP surrogate?*

Table 1: Dataset composition after the progress filter.

Split	Episodes	Failures	Non-failures	Failure rate (%)
Train	4 094	80	3 195	2.4
Validation	819	20	799	2.4
Test	1 734	83	1 651	4.8

4.1 Model Specification and Rationale

To answer our research questions, we compare a LSTM-based surrogate against the baseline MLP surrogate from the Indago framework.

LSTM Surrogate Rationale We choose an LSTM to leverage the temporal data from the DRL agent, which is unused by the baseline model. Our hypothesis is that sequential patterns provide richer information for predicting failures than static initial configurations alone. While we train the LSTM on full interaction sequences, inference is done with only the initial static configuration (as a sequence of length one). This tests whether training on temporal dynamics enables better failure prediction from static initial states. The LSTM architecture is outlined in section 4.4

Baseline MLP Surrogate The baseline for comparison is the MLP surrogate implemented by Biagiola et al. [4]. The MLP is a feedforward architecture with two ReLU-activated layers, trained using a weighted softmax cross-entropy over two classes. We obtained the original model checkpoint directly from the authors. They train the model on the same parking dataset but use only the static features of the initial environment configuration.

4.2 Dataset and Data Transformation

We apply a training progress filter of 50% for the training data and 5% for the test data, consistent with the MLP of Biagiola et al. [4]. This filtering removes early, less relevant episodes from the agent's learning process. The final dataset composition is shown in Table 1.

Input Sequence Representation For the LSTM, we generate the input sequence by evolving the initial environment configuration over time based on the vehicle's trajectory. This ensures that each sequence is a dynamic representation of a single episode, rooted in its initial state. The feature encoding and sequence construction are detailed in Section 3.3.

4.3 Evaluation Protocol

To answer our research questions, we perform two main evaluations:

Internal Predictive Precision (RQ1.1) To assess the LSTM's predictive power, we compute standard classification metrics: precision, recall, F1-score, and AUC [23] as described in Section 3.5. We calculate these on the held-out test set and compare them directly against the performance of the baseline MLP surrogate.

Comparative Evaluation in Indago (RQ1.2) To answer RQ1.2, we evaluate the practical impact of each surrogate model by executing a failure search using the Genetic Algorithm (GA) integrated within the Indago framework [4]. In this setup, the surrogate model acts

as a fitness function, scoring candidate configurations by their predicted failure likelihood. The GA evolves a population, through crossover and mutation, guided by these scores.

The search protocol for this evaluation is as follows: we employ the `sal+fail` strategy, which was demonstrated by Biagiola et al. [4] to be the most effective for the parking dataset. This strategy uses a GA that iteratively evolves a population of environment configurations, guided by the surrogate's failure predictions. The search is initialized with known failure seeds from the training data to provide a strong starting point, and the GA's mutation operator incorporates a saliency-based approach to target the most impactful features. For each GA search run, we set a budget of 3 seconds to find failure-inducing configurations, similar to the MLP's 5 seconds [4]. We run the GA 50 times and evaluate the DRL agent on the top 50 configurations to account for stochasticity in both the GA and the surrogate predictions.

Following the completion of the search runs for both surrogates, we compare their performance based on the following criteria:

- **Number of Failures:** The average count of unique failure-inducing environment configurations discovered by each surrogate guided search. A failure corresponds to an episode ending in either a collision or a timeout, rather than a succesful park.
- **Diversity of Failures:** The diversity of these configurations, measured in both the input space (configuration coverage and entropy) and the output space (DRL agent behavior diversity), using the same clustering-based metrics as the baseline paper [4].

Diversity is essential to ensure comprehensive testing. It reflects the ability to uncover a wide range of failures and behavioral weaknesses, rather than repeatedly exposing similar issues. As Biagiola et al. [4] note, more diverse failures better reveal the robustness of the agent across the full spectrum of the environment space.

To assess whether the observed differences in these metrics are statistically significant, we use the Mann-Whitney U test at a significance level of $\alpha = 0.05$ [19]. Where significance is found, we report the Vargha-Delaney effect size to quantify the magnitude of the difference [26]. We conduct statistical tests by comparing the set of results from multiple seeds for each model.

4.4 Hyperparameter and search settings

Table 2 summarizes the hyperparameter space. We constrain the search to $h \neq f$ to avoid redundancy in feature processing, as it would otherwise project the hidden state into a space of the same dimensionality. We fix the pooling type as *mean pooling*. An `oversample` value of 0.0 uses a weighted softmax cross-entropy rather than oversampling to address class imbalance. The total grid search space consists of 1 512 initial configurations.

We select the hyperparameter configurations through a three-stage setup described in Section 3.5. To narrow the search space, several hyperparameters were fixed across all experiments. Specifically, the surrogate network is a unidirectional LSTM with mean pooling and a ReLU activation function, the optimizer is Adam, and we used a fixed test split of 0.2. For early stopping, we restore the best weights when validation precision had not improved for four evaluations (patience $k = 4$). Since the validation evaluates every 5 training epochs, this corresponds to a precision stagnation of at least 20 training epochs.

Table 2: Hyperparameter search space defined by the Cartesian product. Configurations where $h = f$ are excluded. An oversampling of 0.0 indicates use of a weighted softmax cross-entropy rather than oversampling.

Hyperparameter	Values
n (layers)	$\{1, 2\}$
h (hidden size)	$\{16, 32, 64\}$
f (dense size)	$\{8, 16, 32\}$
p (dropout)	$\{0.3, 0.5\}$
ℓ (learning rate)	$\{10^{-3}, 10^{-4}\}$
`batch_size`	$\{64, 128, 256\}$
`oversample`	$\{0.0, 0.2, 0.4\}$
L (timesteps)	$\{2, 8, 32\}$

Table 3: Hyper-parameters of the top 5 LSTM surrogate models selected from the grid search, ordered by descending validation AUC. We show the model highlighted in gray in the failure search experiments.

Model	Layers	LR	Oversample	Hidden Size	Dropout	Dense Size	Timesteps
1	1	10^{-4}	0.0	64	0.3	32	2
2	1	10^{-4}	0.0	64	0.5	32	2
3	1	10^{-4}	0.0	64	0.3	8	2
4	1	10^{-4}	0.4	64	0.3	32	8
5	1	10^{-4}	0.4	64	0.5	8	2

Table 4: Performance of the top 5 LSTM models on the held-out test set, ordered by descending precision. We show the model highlighted in gray in the failure search experiments.

Model	Precision	Recall	F1 Score	AUC
5	0.115	0.289	0.165	0.653
2	0.094	0.410	0.153	0.659
1	0.093	0.421	0.153	0.659
3	0.090	0.421	0.149	0.654
4	0.083	0.434	0.140	0.597

5 Results

5.1 RQ1.1: Internal Predictive Precision

To answer our first research question, we walk through the top five LSTM surrogate models identified by our multi-stage hyper-parameter search. Table 3 details the hyper-parameter configurations of these models, while Table 4 summarizes their performance on the held-out test set. To assess sensitivity to seed variation, we also evaluate the top 15 configurations across 10 seeds which is summarized in figure 3.

An interesting observation of Table 3 is that the top 5 models do not include a configuration with timesteps = 32. In fact, although not shown here, none of the 15 top models use 32 timesteps while 11/15 of the top models use the 2 timestep configuration.

All top-performing LSTM models exhibit relatively low precision (ranging from 8.3%

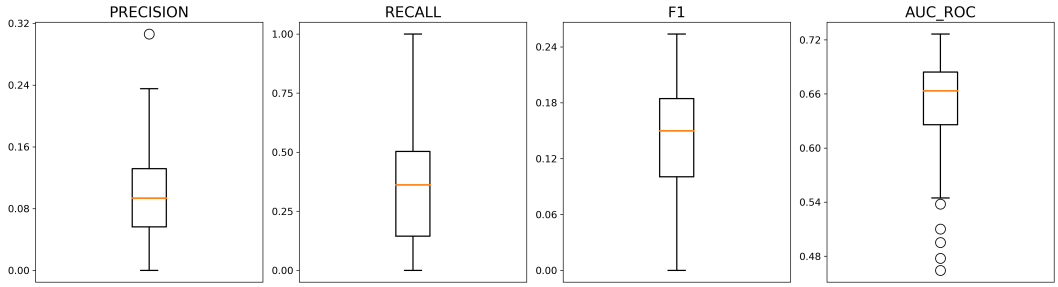


Figure 3: Box-plots of the performance metrics of the top 15 LSTM configurations on a held-out test set over 10 seeds.

to 11.5%) but achieve high recall (ranging from 28.9% to 43.4%). This contrasts with the baseline MLP surrogate from Biagiola et al. [4], which reported a higher precision of 24% but a lower recall of 17% for the same *Parking* environment. As Biagiola et al. [4] argue, high precision should be valuable in this setting, as it ensures the failure search is guided toward configurations that are likely to expose failures. Therefore, the LSTM's lower precision may limit its effectiveness in steering the search efficiently, despite its broader coverage.

In table 4 the AUC scores for the top models are consistently above 0.65 (with the exception of Model 4), indicating that the models have a better-than-random capability to distinguish between failure and success configurations. Model 5 achieves the highest F1-score (0.165), providing the best balance between precision and recall among the candidates.

Figure 3 aggregates the 150 performances on held-out test sets with: (15 configs $\times$ 10 seeds). Precision is right-skewed: the inter-quartile range (IQR) lies between $\approx$0.05 and 0.12, with one outlier at $\approx$ 0.31. Recall spans the full $[0, 1]$ interval, but its long upper whisker seems to correspond to runs that predict failure nearly everywhere (e.g. precision= 0.05, recall= 1.0), so the apparent perfect recall is achieved at the cost of many false positives rather than superior discrimination. Consistently low precision therefore pulls the F1 median toward 0.14, with only occasional rises of precision to 0.25. AUC_ROC is tighter: half of the runs fall in $[0.63, 0.69]$, the median is $\approx$0.66, yet a few seeds dip just below 0.5, signaling that poorly calibrated thresholds can push some models close to random guessing.

Taken together, the box-plots confirm that while ranking ability (AUC) is generally stable, threshold dependent metrics remain vulnerable to both seed choice and the model's tendency toward over-prediction.

5.2 RQ1.2: Comparative Evaluation in Indago

For our second research question, we integrated our LSTM surrogates into the Indago framework to assess their real-world impact on the testing process. To conduct a clear comparison against the baseline, we selected the single best-performing LSTM configuration from our top five candidates. A primary objective of a testing framework like Indago is to maximize the discovery of unique failures. Therefore, we chose the model that was most successful on this metric. Of the five models, Model 5 was the only configuration to find a statistically significant higher number of failures than the MLP baseline. Statistical significance was assessed using the Mann-Whitney U test, and effect size was measured using the Vargha-Delaney statistic.

Table 5: Comparison of the number of discovered failures between the MLP and the best-performing LSTM surrogate (Model 5). Statistical significance was assessed using the Mann-Whitney U test, and effect size was measured using the Vargha-Delaney statistic.

Surrogate	Mean Failures	p-value	Effect Size
MLP (baseline)	14.98	0.0013	Medium
LSTM (ours)	**17.32**		

Table 6: Comparison of diversity for failures discovered by the MLP and best-performing LSTM surrogate. Statistical significance was assessed using the Mann-Whitney U test, and effect size was measured using the Vargha-Delaney statistic.

Metric	MLP (baseline)	LSTM (ours)	p-value	Effect Size
Input Coverage (%)	58.0	**99.0**	$< 10^{-15}$	Large
Input Entropy (%)	8.28	**90.39**	$< 10^{-16}$	Large
Output Coverage (%)	**73.55**	68.97	Not Significant	
Output Entropy (%)	**75.37**	57.59	$< 10^{-5}$	Large

Number of Discovered Failures As shown in Table 5, the LSTM surrogate discovered on average 17.32 failures compared to 14.98 for the MLP, out of 50 configurations. This corresponds to 34.6% and 30% of the total failure space, respectively. This difference is statistically significant ($p < 0.005$). It suggests that training the surrogate on sequential data provides a benefit for maximizing failure discovery. A caveat is that the amount of timesteps used by model 5 during training is only 2. This puts into question the advantage of using the LSTM.

Diversity of Failures Beyond the raw count, we evaluated the diversity of the discovered configurations in both the input space (the environment configurations) and the output space (the DRL agent's resulting behavior).

The results for input diversity are shown in Table 6. The LSTM surrogate outperforms the MLP, achieving nearly 100% coverage of the input failure clusters and an entropy value more than ten times higher. Both differences are statistically significant with large effect sizes. This indicates that the LSTM guided search explores the space of possible environment configurations far more effectively, discovering a much wider and more evenly distributed set of failure-inducing inputs.

The comparison of output diversity in Table 6, reveals a more complex relationship. No significant difference was found in behavioral coverage between the two surrogates. However, the MLP surrogate produced failures with a significantly higher behavioral entropy. This indicates that while the LSTM finds more diverse *inputs*, the failures found by the MLP are more varied in their resulting *behaviors*.

5.3 Discussion of the Results

Our findings reveal a trade-off between a surrogate's predictive metrics and its effectiveness in testing. The LSTM surrogate, despite having lower precision than the MLP baseline, discovered significantly more failures and a dramatically more diverse set of failure-inducing inputs. This suggests that for the purpose of test case generation, a model's ability to be highly accurate on a smaller set of predictions might not be as important as argued by Biagiola et al. [4].

The most interesting insight comes from the fact that most of our best performing models were trained on sequences of only two timesteps. This directly challenges our initial hypothesis that temporal sequences would make for better predictive power.

Training on longer sequences appears to be counterproductive, likely for two related reasons. First, it may introduce a mismatch in training, causing the model to rely on clear failure signals that appear late in a trajectory. These signals are absent during inference, which starts from a single static state, thus degrading performance. Second, it is possible that the benefit of temporal data may lie primarily in the very first steps of an episode.

Ultimately, our findings show that utilizing a small amount of temporal context can improve failure discovery compared to static models. While this does not imply a fundamental architectural advantage of LSTMs over MLPs, it suggests that limited sequential input can provide useful predictive signals. Future work is needed to better understand when such short-term dynamics are most informative and how they interact with the nature of the failures being predicted.

6 Threats to Validity

Internal Validity The performance of deep learning models is sensitive to random initialization. Although we mitigated this by evaluating our top models across 10 seeds, Figure 3 shows that significant performance variance remains. Therefore, the performance of our selected best model (Model 5), while statistically superior to the baseline in our main comparison, could still be influenced by this randomness.

Construct Validity Our choice of Model 5 for the main comparison (RQ2) was based on its superior failure discovery rate, not its AUC score. This prioritizes one metric, and selecting a different model might have altered our conclusions about the trade-offs between failure discovery and diversity. The very definition of "effective testing" is a construct threat. Our conclusion favors the LSTM's strengths (failure count, input diversity), but a different testing objective, such as maximizing unique behaviors (output diversity), might have favored the baseline MLP.

External Validity Our experiments were limited to the *Highway-Env* Parking scenario. While this is a common environment, the findings may not generalize to other environments, such as the others tested by the baseline's authors [4]. Our key finding, that a minimal LSTM sequence length is most effective, challenges the initial hypothesis for using LSTMs. This result may be an artifact of the Parking environment's specific failure modes rather than a generalizable principle.

Conclusion Validity We strengthened our conclusions by addressing statistical threats. First, we evaluated 15 configurations across 10 random seeds to account for training variance (see Figure 3). Second, our main comparison between the best LSTM and the baseline involved 50 repeated experiments, with differences analyzed using rigorous tests (Mann-Whitney U for significance and Vargha-Delaney for effect size) to ensure our claims are statistically sound.

7 Conclusions and Future Work

This work set out to improve the testing of DRL agents by investigating whether a surrogate model capable of processing sequential data could outperform a static one. Our central hypothesis was that an LSTM network, by exploiting temporal information from sequential agent interactions, would provide a more beneficial surrogate for discovering failures using the Indago framework compared to the baseline MLP. While the LSTM surrogate did not surpass the MLP in terms of internal predictive precision, exhibiting lower precision in exchange for higher recall, its practical application within the testing framework proved effective. The best performing LSTM model discovered a higher number of failures compared to the MLP (34.6% vs. 30%), and achieved a wider exploration of the input space, with 99% input coverage and 90.39% entropy versus the MLP's 58% and 8.28%, respectively. However, this came at the cost of reduced output diversity, suggesting that while the LSTM more thoroughly explores the environment space, it may concentrate on a narrower set of failures with respect to behavior.

A notable finding is the unexpected prominence of models trained on sequences of only two timesteps. This outcome challenges our initial hypothesis that temporal contexts would provide improved predictive power. There is a possibility that shorter sequences prevent the model from erroneously relying on late-stage failure signals that are unavailable during inference. This success, therefore, can stem from avoiding a critical mismatch between training and testing conditions.

These findings open several avenues for future research. A main direction would be to further investigate this "short-term dynamics" phenomenon to determine if there is a benefit specific to the first state transition and whether simpler models could capture it more efficiently. Furthermore, the trade-off observed between the LSTM's superior input diversity and the MLP's higher behavioral diversity calls for deeper exploration. A hybrid approach or multi-objective search could potentially combine the strengths of both surrogates. Finally, validating these findings across different DRL agents and environments would be essential to assess the generalizability of using temporal models for failure discovery. Ultimately, while LSTMs are not a universal solution, our results indicate that a minimal temporal context can improve certain aspects of the failure discovery process in DRL testing. This points to a potentially valuable role for short sequence models as a computationally efficient testing tool.

8 Responsible Research

8.1 Ethical Considerations

The study relies solely on synthetic interaction logs generated by Highway-Env [16] and released by Biagiola et al. [4]. No personal data or human annotation is involved, so GDPR or informed-consent issues do not arise. Results are reported for research only. The LSTM surrogate will not be deployed in any safety-critical system. Nevertheless, we note that false-negative predictions could mask hazardous configurations if the method were reused for real vehicles. Because the simulator models a single parking-lot layout, we cannot rule out coverage bias, broader scenarios remain future work. The 1512 point grid search ran for ≈ 20 GPU-hours on an NVIDIA P1000, a modest footprint mitigated by early-stopping and three-stage filtering. We occasionally used generative AI in writing for phrasing improvements and structural suggestions.

8.2 Reproducibility

The full pre-processing pipeline (sequence construction, heading approximation, class-imbalance handling) is specified step-by-step in Section 3. All experiments were conducted on the fixed Parking dataset described in Section 4.2 (Table 1). Model architectures, hyper-parameter grids, and the three-stage evaluation protocol are documented in Section 3.5. Experiments ran on a single HP ZBook (Intel i7-8750H, 16 GB RAM, NVIDIA P1000) with Windows 11, Python 3.9, PyTorch 1.8.0, and CUDA 11.1.
To control stochasticity, we fixed and logged random seeds for:

- all surrogate training and validation runs,
- every replicate in the multi-seed grid-search stages,

Genetic-algorithm (GA) searches intentionally used fresh seeds per trial to mirror the practice of Biagiola et al. [4].

Because every pre-processing step, split, and hyper-parameter value is already enumerated in the paper, an independent researcher can reproduce every figure and table by following those sections, executing the supplied scripts, and running the GA with the logged seeds. Thus the study is fully self-contained and repeatable without hidden dependencies.

The code and data to reproduce the results can be found with the paper at `https://cse3000-research-project.github.io`.

References

[1] Florent Altché and Arnaud de La Fortelle. An lstm network for highway trajectory prediction. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 353–359, 2017.

[2] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.

[3] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.*, 6(1):20â29, June 2004.

[4] 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.

[5] Fabio Cassano, Anna Maria Crespino, Mariangela Lazoi, Giorgia Specchia, and Alessandra Spennato. An ews-lstm-based deep learning early warning system for industrial machine fault prediction. *Applied Sciences*, 15(7), 2025.

[6] Paolo Conti, Mengwu Guo, Andrea Manzoni, and Jan S Hesthaven. Multi-fidelity surrogate modeling using long short-term memory networks. *arXiv [math.NA]*, 2022.

[7] Luigi De Simone, Enzo Caputo, Marcello Cinque, Antonio Galli, Vincenzo Moscato, Stefano Russo, Guido Cesaro, Vincenzo Criscuolo, and Giuseppe Giannini. Lstm-based failure prediction for railway rolling stock equipment. *Expert Systems with Applications*, 222:119767, 2023.

[8] Tolga Ergen, Ali H. Mirza, and Suleyman Serdar Kozat. Energy-efficient lstm networks for online learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):3114–3126, 2020.

[9] Richard Everett. Strategically training and evaluating agents in procedurally generated environments., 2019.

[10] David E Goldberg and John H Holland. *Mach. Learn.*, 3(2/3):95–99, 1988.

[11] Jérémie Guiochet, Mathilde Machin, and Hélène Waeselynck. Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, 94:43–52, 2017.

[12] Arash Hajisharifi, Rahul Halder, Michele Girfoglio, Andrea Beccari, Domenico Bonanni, and Gianluigi Rozza. An lstm-enhanced surrogate model to simulate the dynamics of particle-laden fluid systems. *Computers Fluids*, 280:106361, 2024.

[13] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017.

[14] S Hochreiter and J Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[15] Gary King and Langche Zeng. Logistic regression in rare events data. *Political Analysis*, 9(2):137â163, 2001.

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

[17] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[18] Lei Lin, Weizi Li, Huikun Bi, and Lingqiao Qin. Vehicle trajectory prediction using lstms with spatialâtemporal attention mechanisms. *IEEE Intelligent Transportation Systems Magazine*, 14(2):197–208, 2022.

[19] H B Mann and D R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Stat.*, 18(1):50–60, March 1947.

[20] 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, February 2015.

[21] National Transportation Safety Board. Collision between vehicle controlled by developmental automated driving system and pedestrian, tempe, arizona, march 18 2018. Technical Report HAR-19/03, National Transportation Safety Board, 2019.

[22] Eyder Peralta. Knight capital says it lost $440 million because of computer glitch. NPR, 2012.

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

[24] Richard S. Sutton and Andrew Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts London, England, second edition edition, 2020.

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

[26] András Vargha and Harold D Delaney. A critique and improvement of the CL common language effect size statistics of McGraw and wong. *J. Educ. Behav. Stat.*, 25(2):101–132, June 2000.

[27] 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, 2023.