Reinforcement Learning with Advanced Neural Network Architectures for Test Case Prioritization

Naron Berisha

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Leveraging Convolutional Neural Networks, Dueling Networks, and Transformers for Test Case Prioritization in Continuous Integration Environments

Naron Berisha

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2024

Leveraging Convolutional Neural Networks, Dueling Networks, and Transformers for Test Case Prioritization in Continuous Integration Environments
Naron Berisha

Supervisor: Adrian Eliasson, Plejd AB
Examiner: Kristian Gustafsson, Department of Physics

Master's Thesis 2024
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeXPrinted by Chalmers Reproservice
Gothenburg, Sweden 2024

Reinforcement Learning with Advanced Neural Network Architectures for Test Case Prioritization
Naron Berisha
Department of Physics
Chalmers University of Technology

Abstract

This thesis investigates the application of contemporary neural network architectures, specifically Convolutional Neural Networks (CNNs), Dueling Networks, and Transformers, to enhance Test Case Prioritization (TCP) within Continuous Integration (CI) environments using the RETECS framework and reinforcement learning techniques. Traditional TCP methods often fail to keep pace with the rapid development cycles and complex codebases characteristic of modern software development. By integrating these advanced deep learning techniques within the RETECS framework, this research aims to improve both the efficiency and effectiveness of the TCP process. A rigorous experimental setup, utilizing real-world datasets, was employed to train and evaluate the proposed models against traditional TCP methods. Performance was assessed using key metrics such as the Average Percentage of Faults Detected (APFD) and Normalized APFD (NAPFD), focusing on the models' capability to efficiently detect faults and reduce testing time. Results indicate that the advanced models, especially the Dueling Networks, outperform traditional methods. Specifically, the Dueling Networks demonstrated a consistent increase in APFD scores by up to 3% over conventional models, showcasing their potential to optimize test case scheduling and fault detection within CI pipelines. These findings underscore the transformative potential of neural networks in automating and optimizing TCP processes within CI frameworks, suggesting a shift toward more intelligent testing systems. Moreover, the study highlights the critical role of tailored reward functions in reinforcement learning-based TCP to further enhance the prioritization efficacy of these models. This research contributes to the fields of software engineering and machine learning by providing empirical evidence of the benefits of applying sophisticated computational models to TCP. It also outlines future research directions, including the exploration of additional machine learning models and the integration of these methods into real-world CI environments to further refine and enhance the TCP process.

Keywords: Continuous Integration, Test Case Prioritization, Machine Learning, Neural Networks, Reinforcement Learning, Convolutional Neural Network, Dueling Network, Transformer.

Acknowledgements

I would like to express my sincere gratitude to my examiner, Kristian Gustafsson, for his patience and guidance throughout the course of this thesis. His support in navigating the administrative aspects of the process has been greatly appreciated.

I am also grateful to my supervisor, Adrian Eliasson, for his valuable advice and for providing me with insights into the company's environment. His contributions have been instrumental in contextualizing my research within a practical framework.

Finally, I extend my heartfelt thanks to my family for their unwavering support and encouragement throughout my academic endeavors.

Naron Berisha, Gothenburg, 2024

Contents

List of Abbreviations

APFD Average Percentage of Faults Detected

CI Continuous Integration

CNN Convolutional Neural Network

DQN Deep Q-Network

LTR Learning-to-Rank

MLP Multilayer Perceptron

MSE Mean Squared Error

NAPFD Normalized Average Percentage of Faults Detected

RETECS Reinforcement Learning for Test Case Prioritization

RL Reinforcement Learning

RTL Ranking-to-Learn

TCP Test Case Prioritization

TCS Test Case Selection

TSM Test Suite Minimization

1 Introduction

Continuous Integration (CI) is a pivotal practice in modern software development, aimed at enhancing code quality and accelerating development cycles through frequent integration and immediate feedback [1]. The practice of CI has evolved significantly over the past decade, transitioning from a useful strategy to a critical component of software development in agile environments. This shift underscores the growing complexity of managing and maintaining software quality amid rapid development cycles.

At the heart of CI is regression testing, which ensures that new changes do not adversely affect existing functionalities. Regression testing involves re-executing previously passed test cases on new software versions to detect any unintended consequences of recent code changes [2]. Given the large suite of test cases that can accumulate in sizeable projects, Test Case Prioritization (TCP) becomes essential. TCP strategically orders test cases to maximize the efficiency and effectiveness of the testing process, often focusing on the early detection of faults [3, 4].

Traditional TCP methods, such as heuristic-based and Genetic Algorithms, prioritize test cases based on various criteria, including predicted fault-proneness and code coverage [5]. In particular, Genetic Algorithms have shown significant promise, optimizing test case management effectively by adapting to changes in the codebase dynamically. This adaptability is crucial for maintaining the reliability and quality of evolving software [6]. Extensively applied across numerous studies, these algorithms demonstrate their capability to tackle complex TCP challenges, thereby underscoring the ongoing need for innovative solutions in large-scale software projects [7]. While traditional TCP methods lay a solid foundation, the integration of genetic algorithms presents a more adaptable and potent solution for navigating the complexities of modern software development.

However, as software systems grow in complexity and their operational environments become increasingly dynamic, there arises a need for even more sophisticated methods to manage these challenges and ensure robustness across varied scenarios. In this context, Reinforcement Learning (RL) emerges as a compelling approach. Leveraging RL allows for the development of systems that adapt their testing strategies based on real-time feedback, continuously optimizing the prioritization of test cases to swiftly address the most pressing issues. This shift towards adaptive, learning-based methods represents a significant advancement in test case prioritization, aiming to more effectively meet the demands of contemporary software development processes.

Building on the need for enhanced TCP methods in CI environments, there is growing interest in harnessing advanced machine learning techniques, particularly RL to refine the prioritization process further. Reinforcement Learning for Test Case Prioritization (RETECS) model, introduced by Spieker et al., exemplifies the potential of RL in dynamically adjusting test case prioritization based on historical test execution data, optimizing for minimal feedback time between code commits and notifications of test failures to developers [8]. This model not only showcases the efficacy of RL in honing TCP strategies but also highlights substantial opportunities for innovation and domain-specific adaptations.

Commissioned by Plejd AB, a medium-sized IoT company, this thesis builds upon the foundations of the RETECS model by exploring alternative deep learning architectures, such as Convolutional Neural Networks (CNNs), Dueling networks, and Transformers, for TCP. Plejd AB aims to improve their CI pipeline to enhance software quality and reduce the time required to detect faults. Celebrated for their efficacy in pattern recognition and feature extraction in high-dimensional data spaces, these architectures are well-suited for treating test cases and their attributes as data that encapsulate patterns of failures and successes. This research explores the feasibility and effectiveness of employing the aforementioned neural network models to learn and enhance TCP strategies within CI environments [9, 10, 11].

The effectiveness of the proposed methods will be evaluated using critical metrics such as the Average Percentage of Faults Detected (APFD) and Normalized Average Percentage of Faults Detected (NAPFD). This comparative analysis will not only measure performance against existing RL-based approaches and traditional TCP heuristics—such as random, weighted, and sorting—but also against advanced methods like Bayesian regression [12]. In particular, the DeepOrder model, which utilizes deep learning for TCP in CI, will be examined to assess its ability to surpass the limitations of current methods [13]. This comparative approach will highlight the potential of advanced neural network models to contribute significantly to the development of more efficient, adaptive, and effective testing strategies in CI settings.

This thesis is structured as follows: Section 2 provides a review of the literature, contextualizing the evolution of TCP within CI and highlighting seminal work in the area. Section 3 details the methodology employed to integrate and test the proposed machine learning models. Section 4 presents the results of the empirical tests conducted, and Section 5 discusses these findings in the context of current and future CI practices. Finally, Section 6 concludes the thesis with a summary of the findings, contributions, and recommendations for future research.

2 Problem Framework and Formulation

This chapter outlines the problem framework and introduces the research questions central to this thesis. It begins with an overview of regression testing in CI environments, emphasizing its importance in modern software development. The chapter then explores specific test case management techniques, such as Test Suite Minimization (TSM), Test Case Selection (TCS), and Test Case Prioritization (TCP). Although TSM and TCS are integral to regression testing, the focus of this thesis is on enhancing TCP to improve the efficiency and effectiveness of the testing process within CI environments. Additionally, this chapter formulates the research questions that guide the study, setting the stage for detailed discussions on machine learning fundamentals and their application in TCP, which are covered in subsequent sections.

2.1 Research Questions

The study is driven by the following research questions:

- **RQ1:** What configurations of the proposed models yield optimal performance on the specific dataset?
- **RQ2:** How do the performance metrics of CNN, Dueling Network and Transformer models compare to those of the RETECS models in TCP?
- **RQ3:** How do the proposed neural network models compare with contemporary models in enhancing test case prioritization within CI environments?

2.2 Regression Testing in Continuous Integration Environments

In the fast-paced world of software development, where changes are continually integrated and deployed, regression testing emerges as a cornerstone practice. This type of testing ensures that new code changes harmonize with existing functionalities without disruption. By rerunning previously executed test cases against modified software, regression testing identifies any unintended consequences triggered by recent updates [2]. This practice is not just a routine check but a critical safeguard that captures defects early in the development cycle, thereby facilitating rapid development while ensuring the system's integrity remains intact. In the realm of CI, where developers frequently merge changes into a shared repository, the importance of regression testing is magnified. Each integration can potentially introduce errors that might compromise the stability and reliability of the software. Thus, effective regression testing within CI environments is pivotal, directly impacting the software's stability and reliability [1].

2.2.1 Test Suite Minimization

Test Suite Minimization, or Test Suite Reduction, is a technique in software testing aimed at reducing the number of test cases within a test suite while ensuring that essential coverage requirements are met. This process is particularly crucial in environments where tests need to be performed frequently, such as in CI setups.
The formal problem of TSM is stated as follows [14]:
Given:

- A test suite TS, consisting of test cases $t_1, t_2, \ldots, t_n$,
- A set of test case requirements $r_1, r_2, \ldots, r_m$ that must be satisfied to provide the desired testing coverage of the program,
- Subsets $T_1, T_2, \ldots, T_m$ of TS, one associated with each requirement r_i, such that any one of the test cases t_j belonging to T_i can be used to test r_i.

Problem: Find a representative set of test cases from TS that satisfies all r_i's.
In this context, the test requirements r_i can represent various test-case needs such as source code statements, decisions, definition-use associations, or specification items.
For example:

- **Source Code Statements**: A requirement r_i could be to ensure that a specific line of code or a set of lines are executed by the test cases. The subset T_i would include all test cases that execute this line of code.
- **Decisions**: A requirement r_i might be to test a particular decision point in the code (e.g., an if-else statement). The subset T_i would consist of test cases that traverse this decision point.
- **Definition-Use Associations**: A requirement r_i could be to test the usage of a variable after it has been defined. The subset T_i would include all test cases that cover this variable's definition and subsequent use.
- **Specification Items**: A requirement r_i could relate to a specific functional requirement or user story in the software specification. The subset T_i would consist of test cases that verify this functionality.

The goal of TSM is to identify and eliminate redundant and obsolete test cases, optimizing the test suite for efficiency without compromising the coverage needed to assure software quality. This minimization aids in managing the overhead of maintaining large test suites and decreases the number of test cases that need to be rerun after changes are made to the software [15].

2.2.2 Test Case Selection

Test Case Selection is a targeted approach in software testing designed to identify test cases from an existing suite that are necessary for testing changes in the software. TCS is especially vital in environments where software updates frequently occur, necessitating efficient and effective retesting.
The problem of Test Case Selection can be formulated as follows [16]:
Given:

- A test suite TS, consisting of test cases t_i,
- An original version of the software P,
- A set of changes C made to the original program P,
- A modified version of the software P', which is the result of applying the changes C to the original program P,
- Associations between the test cases in TS and the parts of P they validate, ensuring coverage of specific functionalities or code regions affected by C.

Problem: Identify a subset $T' \subset TS$ where:

- Each test case in T' covers at least one of the changes in C,
- T' is the minimal set of test cases necessary to assure that all modifications in P' are tested effectively.

The selection criteria typically involve a detailed analysis of which parts of the software are affected by the changes C and selecting test cases that specifically address these changes. This process helps in efficiently managing testing resources by focusing on areas of the software that are at risk due to recent modifications, thereby optimizing the testing process and expediting the feedback loop to developers.
TCS is crucial for maintaining high software quality in rapidly evolving software environments. It ensures that testing efforts are precisely focused and resource-efficient, providing critical validation for new features or fixes without the need to execute the entire test suite.

2.2.3 Test Case Prioritization

Test Case Prioritization is a technique used in software testing to order test cases in such a way that those with higher importance are executed earlier. This strategy enhances the effectiveness of the testing process by potentially detecting faults earlier and improving feedback times for development teams.
The objective of TCP can be formally stated as follows [3]:
Given:

- A test suite TS, consisting of test cases t_i,
- A set of criteria based on which the prioritization is to be done, such as likelihood of fault detection, criticality of test cases, or test cost.

Problem: Arrange the test cases in TS in an order TS' such that the test cases that are more likely to detect faults, or are more critical, are executed before those deemed less likely or less critical.
The goal of TCP is to maximize some objective function over the sequence of test cases. Common objectives include:

- Maximizing the rate of fault detection early in the test sequence, measured by metrics such as APFD.
- Minimizing the time until critical faults are detected.
- Balancing the cost of test execution and the benefit derived from early fault detection.

This prioritization not only ensures that critical test cases are executed first but also aims to use the limited testing resources more efficiently. By focusing on the most significant tests earlier, TCP can significantly reduce the cost and time associated with the testing process while still maintaining high software quality.
Test Case Prioritization is especially crucial in CI environments, where frequent integration and testing cycles demand that feedback on new changes is provided as quickly as possible. Proper prioritization ensures that each integration cycle is as informative as possible, aiding in quick identification and resolution of defects.

2.2.4 Metrics for Evaluating TCP

The APFD metric quantitatively evaluates the effectiveness of a test case prioritization in terms of fault detection.
The APFD value is calculated using the formula:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{nm} + \frac{1}{2n} \tag{1}$$

where TF_i is the position of the first test case that detects the i^{th} fault, n is the total number of test cases, and m is the total number of faults. This formula aggregates the weighted average of the percentage of faults detected across the timeline of test case execution.
While APFD provides a measure of prioritization quality, it does not account for the actual execution cost or time of test cases. To address this, the Normalized Average Percentage of Faults Detected (NAPFD) is used.
NAPFD extends APFD by incorporating the ratio of detected to detectable failures within the test suite, making it more suitable for scenarios where not all test cases are executed and some failures might remain undetected. This metric provides a more comprehensive

measure of the testing process's effectiveness by reflecting the efficiency of test case prioritization in real-world scenarios.
The NAPFD is calculated as follows [17]:

$$NAPFD(TS_i) = p - \frac{\sum_{t \in TS_i^{\text{fail}}} \text{rank}(t)}{|TS_i^{\text{fail}}| \times |TS_i|} + \frac{p}{2 \times |TS_i|}$$
$$\text{where p} = \frac{|TS_i^{\text{fail}}|}{|TS_i^{\text{total, fail}}|} \tag{2}$$

Here, TS_i represents the set of test cases, TS_i^{fail} is the subset of test cases that fail, rank(t) denotes the position of test case t within the execution sequence, and $|TS_i^{\text{total, fail}}|$ indicates the total count of detectable failed test cases across all test sequences considered. Detectable failed test cases are those that can be identified by the test suite if executed. This formulation adjusts the traditional APFD metric to reflect the impact of the ratio of detected to detectable failures on the prioritization efficiency, providing a more comprehensive measure of the testing process's effectiveness.
Additionally, the **time scheduling ratio** is introduced as a key metric for evaluating the efficiency of test case execution. This ratio represents the proportion of the total testing time allocated to the execution of prioritized test cases relative to the overall available testing time. It ensures that high-priority test cases are executed within the optimal time frame to maximize early fault detection. For the case of all failures being detected ($p = 1$), the formula reduces to APFD [8].

2.2.5 Example Calculations

A. APFD Example Consider a scenario in a software development cycle where we have a set of test cases and corresponding faults they detect. The objective is to demonstrate how the APFD metric shown in Equation (1) can be calculated to evaluate the effectiveness of a given test case prioritization. This metric is particularly useful for understanding how early in the test sequence faults are detected, which is crucial for timely fixes in the development process.
Table 1 illustrates the prioritized sequence of test cases and the cumulative faults each detects:

Rank	Test Case Order	Faults Detected	Cumulative Faults Detected
1	T4	F2, F5, F6	F2, F5, F6
2	T1	F1, F2	F1, F2, F5, F6
3	T3	F4, F5	F1, F2, F4, F5, F6
4	T2	F3	F1, F2, F3, F4, F5, F6

Table 1: APFD Calculation Example

We can determine the ranks where each fault is first detected:

- Fault 1 by T1 at rank 2
- Fault 2 by T4 at rank 1
- Fault 3 by T2 at rank 4
- Fault 4 by T3 at rank 3
- Fault 5 by T4 at rank 1
- Fault 6 by T4 at rank 1

Thus, the values TF_i are: 2, 1, 4, 3, 1, 1. Plugging these values into Equation (1) gives:

$$APFD = 1 - \frac{2+1+4+3+1+1}{4 \times 6} + \frac{1}{2 \times 4} = 0.625$$

This example clearly demonstrates how the APFD metric can be calculated given a specific prioritized sequence of test cases.

B. NAPFD Example Building on the previous APFD calculation example, consider a scenario where not all faults are detected. In this case, assume that only 5 out of the 6 detectable faults are identified during the test execution.
Using the NAPFD formula in Equation (2) with the following values:
$|TS_i| = 4$ (total number of test cases executed),
$|TS_i^{\text{fail}}| = 5$ (number of faults detected by the test cases executed),
$|TS_i^{\text{total, fail}}| = 6$ (total number of detectable faults).
Thus, $p = \frac{5}{6} \approx 0.833$.
The ranks where each fault is first detected are:

- Fault 1 by T1 at rank 2
- Fault 2 by T4 at rank 1
- Fault 3 by T2 at rank 4
- Fault 4 by T3 at rank 3
- Fault 5 by T4 at rank 1
- Fault 6 by T4 at rank 1 (not detected in this scenario)

The sum of ranks of detected faults is:

$$\sum_{t \in TS_i^{\text{fail}}} \text{rank}(t) = 2 + 1 + 4 + 3 + 1 = 11$$

Substituting the values into the NAPFD formula:

$$\begin{aligned} NAPFD &= p - \frac{\sum_{t \in TS_i^{\text{fail}}} \text{rank}(t)}{|TS_i^{\text{fail}}| \times |TS_i|} + \frac{p}{2 \times |TS_i|} \\ &= 0.833 - \frac{11}{5 \times 4} + \frac{0.833}{2 \times 4} \\ &= 0.833 - \frac{11}{20} + \frac{0.833}{8} \\ &= 0.833 - 0.55 + 0.104 \\ &= 0.387 \end{aligned}$$

So, the NAPFD value for the given test case prioritization example is 0.387, which is different from the APFD value of 0.625. This highlights the impact of considering the ratio of detected to detectable failures in the evaluation of test case prioritization effectiveness.

2.3 Problem Formulation

This thesis adopts and extends the formal definitions and notations outlined in the RETECS study by Spieker et al. (2017), focusing on enhancing TCP within CI environments through a reinforcement learning-based approach. The goal is to maximize the efficacy of detecting faults early in the regression testing cycle, thus reducing overall development time and improving software quality. The key notations used are:

- T_i: A set of test cases $\{t_1, t_2, \ldots, t_N\}$ for each CI cycle i, reflecting the dynamic and evolving nature of test environments.
- TS_i: An ordered subset of T_i selected for execution, representing the test schedule.
- $t.\text{verdict}_i$ and $t.\text{duration}_i$: Properties of each test case in TS_i, known only after execution.
- TS_{fail_i}: The subset of T_i comprising test cases that failed, indicating the need for immediate attention.

The methodologies explored, including Time-limited TCP and Adaptive TCS, aim to construct a test schedule TS_i that maximizes a performance measure $Q_i(TS_i)$, constrained by available execution time and resources. This approach not only seeks to optimize resource allocation but also ensures timely fault detection, which is critical in maintaining the rapid pace of CI environments.

3 Machine Learning Fundamentals

Machine learning is a subset of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed [18]. It encompasses various algorithms and techniques for creating models that can make predictions or decisions based on data.
Recent advancements in machine learning have introduced innovative strategies such as Learning-to-Rank (LTR) and Ranking-to-Learn (RTL), which have been tailored to enhance traditional testing methodologies [19]. These strategies are particularly adept at optimizing test case selection and prioritization by dynamically adapting to changes in the codebase and test suite. This adaptability not only minimizes resource expenditure but also maximizes the early detection of critical issues, thereby streamlining the development process and enhancing software quality. Building upon these developments, this thesis delves into the specific application of reinforcement learning within CI environments. Reinforcement learning is distinctively capable of refining testing strategies continuously, based on real-time feedback from ongoing development cycles. This approach is anticipated to significantly boost the efficiency and effectiveness of regression testing practices. By leveraging reinforcement learning, this research aims to develop methods that not only respond to changes more adeptly but also predict and mitigate potential issues before they affect the software development lifecycle.

3.1 Reinforcement Learning Models

Reinforcement learning (RL) is a branch of machine learning where agents learn to make decisions in an environment to maximize cumulative rewards. It does not require labeled input/output pairs and focuses on balancing exploration of uncharted territory and exploitation of current knowledge. The foundation of RL lies in the Bellman equations, which establish recursive relationships for evaluating and optimizing decision-making strategies [20].

Basic Concepts

- **State** s_t: The environment's state at time t. This represents all relevant information about the environment at a specific point in time.
- **Action** a_t: An action taken by the agent at time t. Actions are decisions made by the agent that influence the state of the environment.
- **Reward** r_t: Reward received after taking action a_t at time t. The reward is a scalar feedback signal indicating the immediate benefit of the taken action.
- **Policy** π: A strategy that the agent employs to decide an action based on the current state. Formally, $\pi(a|s)$ is the probability of taking action a when in state s.
- **State-Action Value Function** $Q(s, a)$: Measures the expected utility of taking an action a in state s and following policy π. This function, also known as the Q-function, represents the expected cumulative reward an agent can obtain by taking action a in state s and then following the policy π thereafter.
- **Next State** s': The subsequent state the environment transitions to after the agent takes action a in state s.
- **Next Action** a': The action taken by the agent in the subsequent state s' according to the policy π.

- **Expected Value** $\mathbb{E}$: Represents the expected value operator, which computes the average outcome of a random variable according to its probability distribution.
- **Discount Factor** γ: A factor $\gamma \in [0, 1)$ that discounts future rewards. It determines the importance of future rewards relative to immediate rewards. A higher γ places more weight on future rewards.
- **Learning Rate** α: A parameter $\alpha \in (0, 1]$ that determines how much new information overrides old information during learning. A higher α allows faster learning from recent experiences, while a lower α results in slower, more stable updates.
- **Immediate Reward** r: The reward received immediately after taking an action a in state s.
- **Bellman Expectation Equation** $Q^\pi(s, a)$: Quantifies the expected return from a state-action pair (s, a) under policy π. It involves the expected reward r, the discount factor γ, the next state s', the next action a', and the policy π.
- **Bellman Optimality Equation** $Q^*(s, a)$: Gives the maximum reward achievable from any state-action pair (s, a). It involves the immediate reward r, the discount factor γ, the next state s', and the optimal action a' that maximizes the expected utility.
- **Optimal Action-Value Function** Q^*: The function $Q^*(s, a)$ that provides the highest expected reward from state s when taking action a and thereafter following the optimal policy.

The Bellman Expectation Equation quantifies the expected return from a state-action pair (s, a) under any policy π as follows:

$$Q^\pi(s, a) = \mathbb{E}\left[r + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a')\right]$$

where r represents the immediate reward, γ is the discount factor that diminishes the weight of future rewards, $\mathbb{E}$ denotes the expected value, and $Q^\pi(s', a')$ evaluates the subsequent state-action pairs. Here, $\pi(a'|s')$ is the policy that determines the probability of taking action a' in state s'.
To find the optimal policy, the Bellman Optimality Equation is used, giving the maximum reward achievable from any state-action pair as:

$$Q^*(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a')\right] \tag{3}$$

This optimal action-value function Q^* is crucial for algorithms that seek to optimize the policy, as it allows for the direct computation of Q^* without explicit knowledge of the policy itself [21].

Deep Q-Network (DQN) integrates this optimal action-value concept with the computational power of deep learning to approximate Q^* using neural networks [22]. DQNs use the following loss function derived from the Bellman Optimality Equation (3) to update the neural network parameters:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim U(D)}\Big[\big(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\big)^2\Big]$$

Here, θ are the parameters of the neural network that approximates the Q-function $Q(s, a)$, and θ^- are the parameters from a previous iteration. These parameters θ^- are updated every c episodes to stabilize learning. This periodic update, denoted by c, is crucial for maintaining stability in the training process [22]. By synchronizing θ^- periodically rather than continuously, the network experiences less volatility in the expected target values during learning, facilitating smoother convergence and reducing the risk of divergence. This periodic update mechanism helps to counteract the problem of moving targets in the training data. Additionally, the replay buffer $U(D)$, which stores transitions, plays a crucial role in mitigating issues of correlation and non-stationarity in the sequential updates [23]. By sampling from this buffer, the DQN algorithm is fed a more diverse set of experiences, further enhancing the robustness and efficacy of the learning process.
For example, a robot learning to navigate through a crowded environment receives rewards for avoiding collisions and reaching its destination efficiently. By applying a Deep Q-Network, the robot adapts its strategy based on the feedback from its interactions with the environment. This approach eliminates the need for a predefined map and demonstrates the effectiveness of integrating reinforcement learning with deep learning to facilitate complex decision-making in dynamic environments [24].

3.2 Deep Q-Networks and Experience Replay

3.2.1 Benefits of DQNs

Deep Q-Networks (DQNs) combine traditional reinforcement learning techniques with the computational power of deep neural networks. This integration allows DQNs to effectively process large and complex state spaces encountered in real-world scenarios, such as video games or robotic navigation. Key benefits of using DQNs include:

A. Handling High-Dimensional State Spaces
Traditional Q-learning employs a tabular approach to store and update Q-values, which becomes impractical

in environments with large state spaces. DQNs address this limitation by using deep neural networks to approximate the Q-value function, significantly reducing the dimensionality and complexity of the problem [22].

B. Generalization DQNs excel in generalizing across similar states, unlike tabular methods that learn the action-value function separately for each state-action pair. This generalization enhances the efficiency and robustness of the learning process, allowing behaviors learned in one state to influence actions in other, unseen states [25, 26].

C. Stability and Convergence DQNs incorporate techniques such as Experience Replay and Fixed Q-Targets to address issues of instability and divergence commonly associated with applying deep learning to reinforcement learning. Experience Replay smooths out the learning distribution by breaking the correlation between consecutive learning samples, while Fixed Q-Targets stabilize training by holding the target network's weights fixed for a number of steps [22].

D. Scalability and Flexibility The scalability of DQNs allows them to be applied across various environments without significant modifications to the architecture. This flexibility is demonstrated in their successful deployment in domains ranging from video games to robotic control tasks, with the potential for adaptations such as integrating convolutional layers for visual input processing or recurrent layers for handling temporal dependencies [22, 27].

E. Improved Learning Efficiency By integrating neural networks, DQNs learn more efficient representations of the environment, leading to faster and more effective policy learning. This efficiency is crucial in complex environments where agents must make decisions based on large amounts of sensory input [22].

3.2.2 Experience Replay

The concept of experience replay is primarily utilized in DQNs, where each experience is stored as a tuple of $(state, action, reward, next\_state)$. The DQN algorithm uses these experiences to repeatedly update the action-value function according to the update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

This update rule effectively captures the dynamic nature of the environment, facilitating continual learning and adaptation. The application of DQNs in scenarios such as autonomous driving or robotics, where environmental conditions are unpredictable and constantly changing, showcases their ability to adjust policies on-the-fly via updated Q-values, ensuring optimal or near-optimal decision-making [22, 25].

Overall, the adaptive learning approach of DQNs not only enhances the performance of agents by refining their decision-making processes but also improves their robustness and reliability in facing real-world challenges. These characteristics underscore the significance of advanced reinforcement learning techniques in the development of intelligent systems capable of autonomous operation under a broad range of conditions [28].

3.3 Reward Functions in Reinforcement Learning for TCP

Reward functions are crucial components in RL, guiding the model by providing feedback on the efficacy of actions taken. In the domain of TCP within CI environments, these functions direct the learning algorithm to prioritize test cases that maximize early fault detection. This study employs the reward functions as specified in the RETECS model: *FailCount*, *TimeRank*, and *TCFail*, noted for their effectiveness in this setting [8].

3.3.1 Failure Count Reward

The Failure Count Reward is designed to enhance the prioritization of test cases that frequently detect faults, thereby improving the detection of regressions early in the test cycle. This reward is given to all test cases irrespective of their execution status, reflecting the total number of failing test cases in the test suite:

$$\text{reward}_i^{\text{fail}}(t) = |TS_i^{\text{fail}}| \qquad (\forall t \in T_i) \tag{4}$$

where $|TS_i^{\text{fail}}|$ denotes the number of failed test cases in the test suite at a given CI cycle.

3.3.2 Test Case Failure Reward

The Test Case Failure Reward, defined as the second reward function in the RETECS model, strategically reinforces the scheduling of failing test cases by directly utilizing the test case's verdict as its reward value. This reward mechanism is designed to enhance the prioritization of test cases that detect faults, thereby improving the effectiveness of the test schedule in early fault detection:

$$\text{reward}_i^{tcfail}(t) = \begin{cases} 1 - t.\text{verdict}_i & \text{if } t \in TS_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $t.\text{verdict}_i$ is the outcome of test case t in CI cycle i, indicating whether a fault was detected (0) or not (1).

In this function, failing test cases receive a reward based on their verdict, with a higher reward for detecting faults. No specific reward is given for passing test

cases as their inclusion does not necessarily enhance nor detract from the overall quality of the schedule based on the available information. Importantly, while the order of test cases is not explicitly factored into this reward function, it is implicitly addressed through the system's prioritization logic. By focusing on failing test cases and positioning them earlier in the test sequence, the scheduling method inherently promotes an earlier execution of these critical test cases, as described in Section 3.2 of the RETECS model. This approach not only reinforces the identification and prioritization of problem areas within the software but also aligns with the overarching goals of CI environments to rapidly detect and address issues.

3.3.3 Time-ranked Reward

The Time-ranked Reward function assigns higher rewards to test cases based on their execution order in the testing sequence, promoting a test schedule that prioritizes failing test cases earlier:

$$\text{reward}_i^{\text{time}}(t) = |TS_i^{\text{fail}}| - t.\text{verdict}_i \times \sum_{\substack{t_k \in TS_i^{\text{fail}} \wedge \\ \text{rank}(t) < \text{rank}(t_k)}} 1 \quad (6)$$

This formulation rewards test cases that fail and are scheduled before other failing test cases, thereby enhancing the efficiency of the testing process by reducing the feedback time.
These reward functions, as directly adopted from the RETECS model, are integrated into our RL framework to train advanced neural network models. This adoption ensures consistency with validated methods in the field, allowing comparisons between the RETECS models and the advanced neural networks examined in this paper.

3.4 Incremental Averaging Q-Learning Algorithm

The Incremental Averaging Q-Learning algorithm, as implemented in the Tableau agent, is a variant of the standard Q-learning algorithm. In this variant, the Q-value update is based on incremental averaging rather than the standard Q-learning update rule. The Q-value is updated using the following formula [20]:

$$Q(s,a) \leftarrow Q(s,a) + \frac{1}{N(s,a)} [r - Q(s,a)] \quad (7)$$

where:

- $Q(s,a)$ is the Q-value representing the expected utility of taking action a in state s.
- $N(s,a)$ is the visit count representing the number of times action a has been taken in state s.
- r is the reward received after taking action a.

In this approach, the Q-value is updated incrementally based on the average of the received rewards, which helps in stabilizing the learning process, especially when the state-action pairs are visited multiple times.

3.5 Multilayer Perceptrons

Multilayer Perceptron (MLP) are a class of feedforward artificial neural networks that consist of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each node (except for the input nodes) is a neuron that uses a nonlinear activation function. MLPs are capable of modeling complex relationships in data by learning from examples and are widely used in various machine learning tasks. The architecture of an MLP is as follows [29]:

- **Input Layer:** This layer consists of nodes representing the input features. Each node in the input layer corresponds to a feature in the dataset.
- **Hidden Layers:** One or more hidden layers process the inputs from the input layer. Each node in the hidden layer is connected to every node in the previous layer and uses an activation function to transform the input.
- **Output Layer:** The final layer produces the output of the network. For classification tasks, the output layer typically uses a softmax activation function to provide class probabilities.

The forward propagation in an MLP can be described as follows:

$$a^{(l)} = \sigma(W^{(l-1)} a^{(l-1)} + b^{(l-1)})$$

where $a^{(l)}$ represents the activation of layer l, $W^{(l-1)}$ is the weight matrix connecting layer $l-1$ to layer l, $b^{(l-1)}$ is the bias vector, and σ is the activation function.
MLPs are trained using backpropagation, where the error between the predicted output and the actual target is minimized using gradient descent. The gradients of the loss function with respect to the weights and biases are computed and used to update the parameters, allowing the network to learn from the training data.
MLPs are versatile and can be applied to a wide range of tasks, including classification, regression, and function approximation. They form the foundation of many advanced neural network architectures and are a crucial component in the field of machine learning.

3.6 Convolutional Neural Networks

Convolutional Neural Networks are a specialized type of neural networks optimized for analyzing visual data. They are particularly effective in tasks that require recognition and categorization of images due to their

ability to process data with a grid-like topology through the use of convolution operations [30, 9].
CNNs utilize convolutional layers that apply a set of learnable filters to the input image. Each filter, also known as a kernel, moves across the image width and height, performing element-wise multiplication and summing the results. In this study, a 1D CNN is utilized to prioritize test cases. The convolution operation in 1D CNNs can be described mathematically as follows:

$$S(i) = (I * K)(i) = \sum_{m} I(i+m) \cdot K(m) \tag{8}$$

where I is the input signal, K is the convolution kernel/filter, and $S(i)$ represents the result of the convolution operation at index i.
Following the convolutional layers, CNNs typically include [31]:

- **Pooling layers**, which reduce the spatial dimensions (length) of the input signal for the next convolution layer. The most common pooling operation is 1D max pooling, where the maximum element is selected from the region covered by the filter, effectively downsampling the feature map:

$$P(i) = \max_{m \in \text{Region}} S(m)$$

- **Fully connected layers**, which compute the class scores from the features extracted by the convolutional and pooling layers. In a fully connected layer, every neuron in the previous layer is connected to each neuron in the next layer. The output is computed as:

$$y = \sigma(Wx + b) \tag{9}$$

where x is the input vector, W is the weight matrix, b is the bias vector, and σ is the activation function.
CNNs can also be utilized as function approximators in RL settings, particularly useful for approximating the state-action value function, $Q(s, a)$, in environments with large input spaces. This ability allows for effective management of the extensive state spaces typical in applications like TCP, where each state can represent a different configuration of software tests and code modifications [22]. The combination of CNNs with RL has proven effective in handling high-dimensional sensory data, providing a robust framework for decision-making in dynamic and visually-rich environments.

3.7 Dueling Network Architectures for Deep Reinforcement Learning

The dueling network architecture, proposed by Wang et al. [10], represents a significant advancement in model-free RL by introducing a novel structural separation between the state value function and the action advantage function. Unlike traditional RL architectures such as DQNs that integrate the assessment of state values and action advantages into a single estimation stream, the dueling architecture separates these into two distinct streams. This separation provides a more nuanced representation of the environment's dynamics, enhancing both policy evaluation and decision-making processes in environments where actions yield similar results.
Enhanced Learning of State Values: The architecture isolates the state value function $V(s)$, which focuses on the value of being in a given state regardless of the action taken. This dedicated focus can lead to more precise estimations of state values, crucial in scenarios where future rewards depend significantly on the resultant states rather than the immediate actions.
Dedicated Action Advantage Stream: The action advantage stream $A(s, a)$ computes the relative benefits of each action independently. This capability is particularly beneficial in complex environments with many similarly valued actions, allowing the model to identify subtle differences in action outcomes that might be obscured in a combined approach.
Implications for Policy Evaluation: By decoupling state value and action advantages, the dueling architecture prioritizes understanding what attributes make some states more valuable than others, beyond mere action selection. This enhancement improves policy evaluation by offering a clearer insight into how actions contribute to overall strategy and increases the robustness of the learning algorithm against environmental stochasticity.
Technical Implementation: The architecture modifies the final fully connected layer of a standard deep Q-network to support dual streams, whose outputs are aggregated as follows:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right) \tag{10}$$

where $V(s)$ estimates the value of being in state s, and $A(s, a)$ represents the advantage of taking action a in state s. The term $\frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')$ is the average advantage taken over all possible actions from the state s, which is subtracted to normalize the advantages, thus stabilizing the training process.

Advantages of Dueling Architecture

- **Learning Efficiency:** By decomposing the value and advantage functions, the network can learn to evaluate the state value without having to learn the effects of each action in every state. This is particularly effective in environments where many actions share similar values.

- **Generalization:** The separate treatment of state value and advantages allows the architecture to generalize better across actions that do not affect immediate rewards, thereby facilitating more robust policy evaluation.
- **Stability:** The subtraction of the average advantage in the output layer reduces variance in the estimated function, enhancing training stability.

3.8 Transformers in Machine Learning

Transformers have revolutionized the field of natural language processing and are increasingly being adapted for various other domains of machine learning. Introduced by Vaswani et al. in their seminal paper "Attention is All You Need" [11], the Transformer model is based on a self-attention mechanism that directly computes relationships between all tokens in a sequence, regardless of their positions. The core innovation of the Transformer is the self-attention mechanism, which allows the model to weigh the significance of different tokens within the input data relative to each other. The architecture of a Transformer is typically composed of an encoder and a decoder:

- **Encoder:** The encoder reads and processes the input sequence. It consists of a stack of identical layers, each containing two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network.
- **Decoder:** The decoder is responsible for generating the output sequence and also consists of a stack of identical layers. In addition to the two sub-layers present in the encoder, each decoder layer has a third sub-layer that performs multi-head attention over the encoder's output.

The self-attention mechanism allows the model to consider every part of the input sequence when processing a particular part of the sequence. This is beneficial for tasks where context is crucial. The attention function can be described as mapping a query and a set of key-value pairs to an output, where the output is a weighted sum of the values, weighted by the computed relevance of the query to each key [11]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q, K, and V represent queries, keys, and values respectively, and d_k is the dimensionality of the keys. Transformers have the potential to be particularly effective in TCP within CI environments, not only due to their ability to handle sequences but also because of their flexibility in processing individual elements within these sequences. In the typical application of transformers, entire sequences are processed to capture complex dependencies. However, in our approach, we adapt the transformer model to focus on individual test cases as they arrive. This adaptation allows the transformer to continuously update its understanding of the test environment based on each new test case, without the need for batch processing the entire test suite.

4 Methods

This chapter outlines the methodological framework employed to explore and evaluate various neural network architectures for TCP within a CI environment at a medium-sized software company. The methodology is designed to address the research questions posed in Section 2.1, focusing on the performance of different deep neural network models in improving the prioritization process.

4.1 Research Design

The research was conducted using an exploratory and iterative design, allowing for the systematic exploration of different model architectures and datasets. This approach enabled the adaptation of models based on continuous feedback and evolving requirements from the company. The first phase involved setting up the RETECS simulation environment, which serves as a benchmark for testing various TCP strategies. The objective was to ensure the environment was correctly configured to replicate previous studies and provide a baseline for further experimentation. The RETECS framework was configured to specifically simulate a CI environment using industrial datasets, which are crucial for evaluating the effectiveness of neural network models in prioritizing test cases. This setup involved scenarios based on actual data from three industrial sources, including two from external companies and one from the company hosting this thesis project. Regular interactions with the company were crucial in shaping the research direction and methodology. Through periodic meetings and a workshop, requirements were gathered and refined, ensuring that the models developed were well-aligned with the company's needs and the practical realities of their CI processes.

4.2 Datasets

This research employs a mixed dataset approach, integrating data from both proprietary industrial sources and open-source projects. ensures a robust evaluation of the machine learning models across diverse environments and test conditions, aligning with the RETECS framework to enable effective comparisons. The datasets used in this study include historical test execution data that captures outcomes and execution times, crucial for training the predictive models. The following datasets are specifically utilized:

- **Paint Control:** Sourced from ABB Robotics, this dataset consists of 114 test cases across 312 CI cycles. It features a total of 25,594 verdicts with a fail percentage of 19.36%. This dataset is indicative of a high-maintenance industrial environment where frequent updates and checks are common.
- **IOF/ROL:** Also from ABB Robotics, this larger dataset includes 2,086 test cases conducted over 320 CI cycles. It contains 30,319 verdicts, with a higher fail percentage of 28.43%, suggesting a complex or less stable system that benefits significantly from intensive testing.
- **Plejd:** Provided by Plejd AB, this dataset involves 457 test cases run through 423 CI cycles, accumulating 92,969 verdicts with a fail percentage of 10.45%. This dataset reflects a moderately stable environment with an intermediate level of fault occurrence.

Data from each source is preprocessed to align with the RETECS framework's requirements. This involves structuring each test case's data to include not only the outcome and execution time but also other relevant metadata that might influence the prioritization process. The preprocessing steps ensure that the data is consistent and comparable across different models and frameworks. The datasets are integrated into the RETECS simulation environment, which replicates a CI pipeline with automated test case execution and prioritization. This integration is crucial for evaluating the real-world applicability of the proposed machine learning models and comparing their performance against traditional methods. The structured and processed data feeds directly into the training and evaluation of the neural network models, ensuring that each model is tested under conditions that closely mimic their intended operational environments. See table 2.

4.3 Input Features and Preprocessing

To effectively train the neural network models for TCP, the input data is preprocessed to capture essential features that reflect the state of the test cases and their execution history. The preprocessing function standardizes the input features to ensure consistency and relevance.

A. Scenario Metadata: Scenario metadata encompasses additional information about the testing scenario that is critical for preprocessing. This includes:

- **maxExecTime:** The maximum execution time of any test case in the scenario.
- **minExecTime:** The minimum execution time of any test case in the scenario.
- **totalTime:** The total execution time of all test cases combined.

These parameters provide context for normalizing the execution times and understanding the range of test case durations within the scenario.

Dataset	Source	Test Cases	CI Cycles	Verdicts	Fail Percentage
Paint Control	ABB Robotics	114	312	25,594	19.36%
IOF/ROL	ABB Robotics	2,086	320	30,319	28.43%
Plejd	Plejd AB	457	423	92,969	10.45%

Table 2: Overview of Industrial Datasets Used

B. Preprocessing Function: The preprocess function takes the current state of a test case, relevant scenario metadata, and the specified history length as inputs. It generates a tuple of features that represent the test case's execution characteristics and historical performance.

C. Input Features: The key input features generated by the preprocessing function are as follows:

- **Normalized Duration:** This feature represents the duration of the test case normalized by the total execution time of all test cases in the scenario. It provides a relative measure of the test case's execution time within the context of the entire test suite.
- **Time Since Last Run:** This feature is calculated as the normalized difference between the maximum and minimum execution times in the scenario and the time elapsed since the test case was last run. If the execution time range is zero (i.e., all test cases have the same execution time), this feature is set to zero.
- **Historical Results:** A binary-encoded vector representing the outcomes of the test case's recent runs. A '1' indicates a failure, and a '0' indicates a pass. The length of this vector is determined by the specified history length (`histlen`). If the number of historical results is less than the history length, the vector is padded with '1's to maintain a consistent length. For example, if a test case failed twice and then passed, its history with a `histlen` of 3 would be (1, 1, 0).

4.4 Model Development and Selection

Multiple deep learning architectures were considered to identify the most effective models for TCP. This selection process involved developing and testing several models, each designed to accommodate the specific characteristics of the RETECS framework used in this study. The default parameters chosen for the experiments are listed in Appendix A, Table 4. Unless otherwise specified, these are the parameters used across all experiments.
In addition, the MLP architecture introduced in Section 3.5, already implemented in the RETECS paper, was used for comparison to our models. This inclusion allows for a benchmark against an established method within the same framework. The Tableau agent was also utilized, which employs an Incremental Averaging Q-Learning algorithm introduced in Section 3.4 using Equation (7) and is also implemented in the RETECS model, to compare the performance of our models.

A. Model Architectures and chosen Parameters: Three distinct neural network architectures were selected based on their potential to enhance TCP performance: CNN (Section 3.6), Dueling Network (Section 3.7), and Transformer (Section 3.8).

- **Hidden Layer Size:** This parameter defines the number of nodes in each hidden layer, determining the complexity of the network's architecture and its capacity to learn intricate patterns from the data.
- **History Length:** This parameter specifies how far back the model looks into the test case failure history, capturing the extent of historical context considered during decision-making.

B. Specific Configurations: Each model employs a tailored configuration to best utilize its architectural strengths:

- **MLP:** The Multi-Layer Perceptron model is designed with one or more hidden layers of fully connected neurons, see Equation 9. Each neuron in a layer receives input from all neurons of the previous layer, enabling the model to learn complex patterns. The MLP uses the ReLU activation function for non-linearity, and its training is optimized using the Adam optimizer. The model can be configured as either an MLPClassifier or an MLPRegressor depending on the output requirements, making it adaptable for both classification and regression tasks in TCP.
- **CNN:** Optimized for sequential data processing, this model uses a convolutional layer followed by dropout and pooling layers to efficiently capture spatial dependencies within input data.
- **Dueling Network:** Employing a dueling architecture, this model separates the estimation of state values and the advantages of actions, facilitating nuanced decision-making that balances immediate and long-term benefits.
- **Transformer:** This model utilizes self-attention mechanisms, with multiple heads and encoder-decoder layers, to handle the sequential nature of test cases and manage complex data patterns across different scales.

C. Training and Optimization: Each model was subjected to a rigorous training regimen using the Adam optimizer, known for its efficiency in handling sparse gradients and adapting learning rates [32]. The training process involved several epochs where models learned from batch-processed data, aiming to minimize a Mean Squared Error (MSE) loss function. This setup minimizes the squared differences between the predicted outcomes and the actual rewards received from the environment, effectively reducing the prediction error. Regular checkpoints and evaluations ensured that the models did not overfit or diverge in performance.
This methodical approach to model development and selection ensures that each neural network is optimally configured and trained to address the complex requirements of TCP within a CI environment, considering both the data characteristics and the operational context.

4.5 Implementation Details

This subsection outlines the implementation specifics of three advanced neural network models—CNN, Dueling Network, and Transformer—designed for TCP within a CI environment. These models are integral to a comprehensive effort to harness deep learning techniques to improve the efficiency and effectiveness of software testing processes at a medium-sized software company. Each model, implemented in PyTorch [33], has been tailored to capture different aspects of the TCP challenge, from recognizing patterns in test execution data to dynamically adjusting test schedules based on predictive insights.
All models employ online learning through incremental training, enabling them to update with one sequence or batch at a time. This approach allows the models to continuously refine their predictions based on real-time data without requiring the entire dataset, thereby enhancing their adaptability in dynamic CI environments. The training function leverages mini-batches or a 'partial fit'-like approach to support incremental updates effectively.
To further enhance the learning process, our implementation incorporates experience replay (Section 3.2.2). Experience replay involves storing past interactions in a memory buffer and using this data to train the model. By sampling mini-batches of experiences randomly from the replay buffer, the models can learn from a more diverse set of scenarios, breaking the correlation between consecutive experiences and thus smoothing the learning curve. This technique is particularly beneficial in preventing the model from overfitting to recent experiences and helps in achieving more stable and generalized performance.

A. CNN Architecture The CNN model designed for test case prioritization leverages convolutional neural networks' strength in handling sequential data typical of software test cases. This model integrates various architectural features to effectively process and learn from test case data, which includes both categorical and continuous input types.
The model architecture begins with an input layer that receives data transformed to fit the model's requirements, focusing on test case duration and historical results. The convolution operation in 1D CNNs can be described mathematically using Equation (8). These inputs are preprocessed to create a uniform input format where historical outcomes are encoded as binary vectors representing pass or fail statuses.
Following the input layer, the model features a convolutional layer with kernels of size three. This layer is pivotal in identifying patterns within the sequence of test case results, such as recurring failures that might indicate deeper issues within the software being tested. To introduce non-linearity and enhance the model's ability to learn complex and non-linear relationships in the data, the ReLU activation function is applied after convolution operations.
The architecture also incorporates a max pooling layer to reduce the dimensionality of the data, improving the computational efficiency and helping to mitigate overfitting, this operation is defined mathematically in Equation (3.6). To further prevent overfitting, a dropout mechanism is employed with a dropout rate of 30%, randomly omitting a subset of features during the training phase.
For the output processing, the model utilizes a sigmoid activation function in the final layer to convert the learned features into a probability score that estimates the likelihood of test case failure. This output provides a probabilistic assessment that is crucial for prioritizing test cases based on their potential to uncover significant issues in the software.
Optimization of the CNN is carried out using the Adam optimizer, chosen for its effectiveness in managing sparse gradients and adapting learning rates, set here at 0.0009. The loss function used is the MSE, which quantifies the discrepancy between the predicted outcomes and the actual results. The training process is structured around multiple epochs where the model weights are iteratively adjusted to minimize this loss, ensuring that the model gradually improves its predictions and adapts to new data effectively.
Regular evaluations during training help monitor the model's performance and ensure it does not diverge from expected outcomes or overfit to the training data. These checkpoints, using a set of validation data, are essential for maintaining the model's reliability and ensuring its applicability to real-world CI environments.

B. Dueling Network Architecture The Dueling Network architecture is designed to improve decision-making in environments such as test case prioritization by separately assessing the value of states and the advantages of actions. This architectural distinction is beneficial in CI environments, where rapid and accurate test case decisions can significantly influence software development cycles.

The network begins with a convolutional layer that processes the input data to capture increasingly abstract features. This layer uses a kernel size of 3 and padding of 1, maintaining the spatial dimensions of the input, which is crucial for preserving contextual information in the input data. The choice of kernel size was based on empirical testing of various sizes (2 to 5), where a kernel size of 3 provided a balanced performance without significant advantages for larger or smaller sizes. After extracting features through the convolutional layer, the data is flattened and passed through a fully connected layer that prepares it for the dueling operations. Here, the network splits into two distinct pathways:

- The *value stream*, consisting of a single output neuron that estimates the overall value $V(s)$ of the current state s, reflecting the intrinsic desirability of the state regardless of the actions taken.
- The *advantage stream*, which computes a separate advantage $A(s, a)$ for each possible action a from the current state, indicating the relative benefit of each action compared to others.

The final output of the network combines these two streams to produce the state-action values $Q(s, a)$ for each action and is computed using Equation (10). This combination approach ensures that the advantages are zero-centered, which stabilizes the learning process by keeping the advantage scores relative rather than absolute, thus improving training efficiency. Equation (3) is then used to estimate the maximum reward achievable from each state-action pair, providing a reliable basis for training the dueling network architecture.

The network is trained using the Adam optimizer with a learning rate of 0.0009, which is well-suited for handling sparse gradients and adapting the learning rate. The MSE loss function is employed to quantify the prediction accuracy, facilitating effective backward propagation and weight adjustments during training. By iteratively reducing prediction errors, the Dueling Network architecture dynamically refines the prioritization of test cases, integrating effectively into the CI pipeline to enhance the software testing process.

C. Transformer Model Architecture The following model is designed to address the nuances of TCP by processing test case data, encompassing both categorical and continuous inputs. The Transformer architecture incorporates an initial embedding layer that transforms the state inputs—comprising execution times and historical test outcomes—into a dense vector representation. This embedding facilitates the complex data manipulations required for the subsequent layers. The core of the Transformer model is made up of several transformer blocks, each consisting of multi-headed self-attention mechanisms and position-wise feedforward networks. These layers are crafted to analyze the input sequences independently, allowing the model to identify intricate patterns and dependencies across the input data, which is critical for predicting the urgency and potential failure of test cases.

Outputs from the transformer blocks are then channeled through a linear layer, which is coupled with a sigmoid activation function. This setup converts the raw output into a probabilistic score that estimates the likelihood of failure for each test case in upcoming runs, serving as a crucial metric for prioritization.

4.6 Integration into CI Pipeline

The integration of the machine learning models into the existing CI pipeline bridges the theoretical aspects of model development with their practical applications, ensuring that the models not only support but enhance the software testing process at a medium-sized software company. The deployment involves setting up the models to evaluate incoming test cases and generate prioritized lists that are directly fed into the test execution queue. This process is illustrated in Figure 1, which shows the complete workflow from test case input to the execution and feedback stages, highlighting how the models interact with different components of the CI pipeline.

Once deployed, the models operate continuously, analyzing new data, adjusting to changes in project dynamics, and improving over time through a robust feedback loop. This loop is essential for refining the models' predictive accuracies, as it uses the outcomes of test executions and developer feedback to inform subsequent iterations of model training. This feedback mechanism ensures that the models remain adaptive and responsive to the shifting contexts of the development environment and the evolving requirements of the software being tested.

Moreover, the integration strategy emphasizes minimal manual intervention, allowing for seamless operations within the CI pipeline. The models are designed to automatically schedule and execute tests based on their prioritization, with the effectiveness of this prioritization continually assessed through regular evaluations. These evaluations help pinpoint potential enhancements in both the model configurations and the overarching test strategies, facilitating a cycle of continuous improvement that keeps the testing process aligned with best practices and technological advancements.

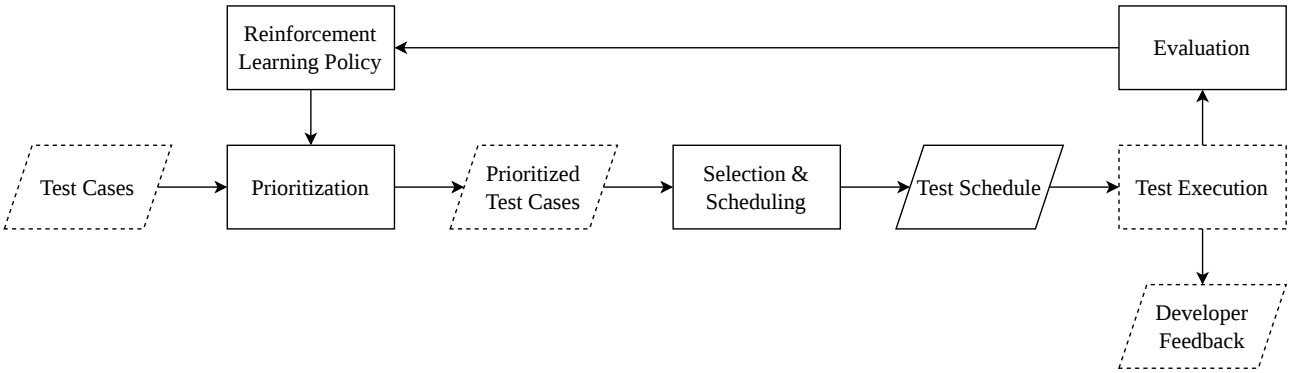

Figure 1: Illustration of the integration of machine learning models into the CI pipeline, detailing the process flow from test case input through prioritization, scheduling, execution, and feedback. This figure demonstrates how the models interact with and enhance the existing CI systems to optimize test case prioritization and execution, based on Fig. 2 in Ref. [8].

To provide a clearer understanding, a schematic illustration of the machine learning implementation can be described as follows:

1. **Data Preprocessing**: Raw test execution data is preprocessed to generate state vectors representing each test case.
2. **State Input**: The state vectors are fed into the neural network model.
3. **Action Output**: The model outputs a single value for each test case, representing its prioritization score.
4. **Execution in CI Pipeline**: The test cases are executed in the CI pipeline based on their prioritization scores.
5. **Reward Calculation**: The outcomes of the test executions are used to calculate rewards.
6. **Experience Replay**: The state-action-reward tuples are stored in the replay memory and used for training the model through incremental updates.

5 Results

This section presents the empirical findings from the experiments conducted to evaluate the performance of the various machine learning models applied to TCP across different datasets. The results focus on the impact of hidden layer sizes, history lengths and reward functions on the models' ability to prioritize test cases effectively. The following data is obtained by averaging the APFD scores over 20 iterations.

5.1 RQ1: Optimal Model Configurations

To address **RQ1**: *What configurations of these advanced models yield optimal performance on the specific dataset?*, this section examines the influence of varying hidden layer sizes and history lengths across the CNN, Dueling Network, and Transformer models. The MLP and Tableau agents are also included for comparison purposes. The reward function used for all subsequent results is Test Case Failure, which is introduced in Section 3.3.2 using Equation (5). The analysis specifically investigates how the number of nodes in the hidden layers for each neural network model affects their performance, as detailed in Section 4.5.

5.1.1 Influence of Hidden Layer Sizes

As discussed in Section 4.5, each neural network model's architecture was designed to capture the nuances of TCP tasks. This analysis specifically investigates how the number of nodes in the hidden layers for each model affects their performance across three datasets: IOF/ROL, Paint Control, and Plejd. By varying the hidden layer sizes, the aim is to identify the configurations that maximize the efficiency and fault detection capability of each model.

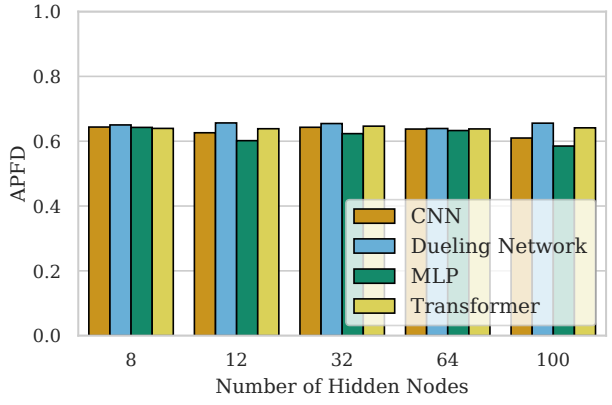

Figure 2: Variation in hidden layer sizes and the corresponding model performance on the IOF/ROL dataset.

The investigation of model performance within the IOF/ROL dataset is presented in Figure 2. The data indicates that for the CNN, Dueling Network, MLP, and Transformer models, variations in the number of hidden nodes, ranging from 8 to 100, do not result in substantial differences in APFD scores. The performance across models remains consistent regardless of the increased hidden layer sizes. This suggests a level of model capacity adequacy for the dataset, as further complexity in the form of additional hidden nodes does not enhance performance metrics.

Figure 3 conveys the APFD scores for the CNN, Dueling Network, MLP, and Transformer models within the Paint Control dataset, considering different sizes of hidden layers. For the CNN and Transformer models, the APFD scores display uniformity, with negligible changes in response to varying numbers of hidden nodes. The Dueling Network model's performance also shows a consistent trend across the range of hidden nodes. However, the MLP model exhibits some fluctuations in APFD scores for this dataset using 12 and 100 hidden nodes, indicating that its performance is somewhat sensitive to the number of hidden nodes used. Despite these fluctuations, the overall trend for all models does not suggest a significant impact of hidden layer size on performance within the tested range.

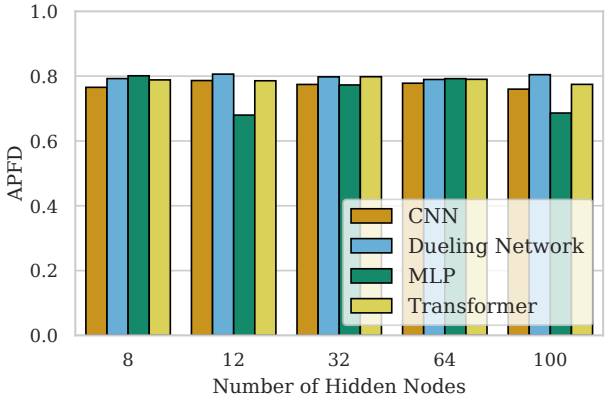

Figure 3: Effect of hidden layer sizes on model performance on the Paint Control dataset.

Figure 4 illustrates the APFD scores for the Plejd dataset across varying hidden layer sizes. For the Dueling Network and Transformer models, the scores remain consistent, showing no substantial improvement with an increase in the number of hidden nodes. The performance of the CNN model similarly exhibits stability across the tested range. In contrast, the APFD scores for the MLP model are notably lower at hidden layer sizes of 12 and 100 nodes, suggesting a decrease in performance at these specific configurations.

Overall, the analysis across all three datasets indicates that varying the hidden layer size does not significantly

affect the performance of most models. The CNN, Dueling Network, and Transformer models exhibit consistent APFD scores regardless of the hidden layer sizes. However, the MLP model shows some sensitivity to hidden layer size, with performance fluctuations observed in the ABB Paint Control and Plejd datasets. The findings suggest that the models generally possess adequate capacity for the datasets, and further complexity in the form of additional hidden nodes does not markedly enhance prioritization performance.

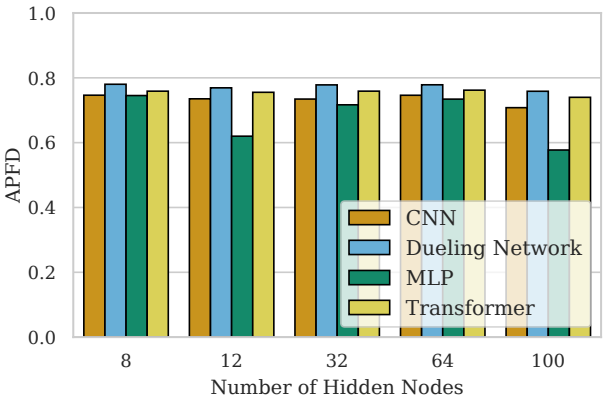

Figure 4: Influence of different hidden layer sizes on the performance of models on the Plejd dataset.

5.1.2 Effect of History Length

The influence of history length on model performance was analyzed by focusing on the stability and variability of APFD scores across different agents. As illustrated in Figure 5, the CNN agent maintains consistent APFD scores across varying history lengths, demonstrating robust stability despite changes in input sequence length. In contrast, the Dueling network agent shows variability in performance, achieving a peak APFD score of 82% at a history length of 25, suggesting this as the optimal length for this model beyond which no further improvements are noted. The Transformer agent's performance is relatively stable across different history lengths, highlighting its effectiveness in managing various temporal depths without significant fluctuations in APFD scores. Performance variability is more pronounced with the Tableau agent, which exhibits greater fluctuations in APFD scores compared to the other agents, indicating a potential sensitivity to the length of input history. The MLP agent experiences a decline in APFD scores as history length increases, which suggests challenges in processing longer input sequences effectively on this dataset.

As shown in Figure 6, the Tableau agent's performance diminishes with an increase in history length, consistently being outperformed by all other agents at every

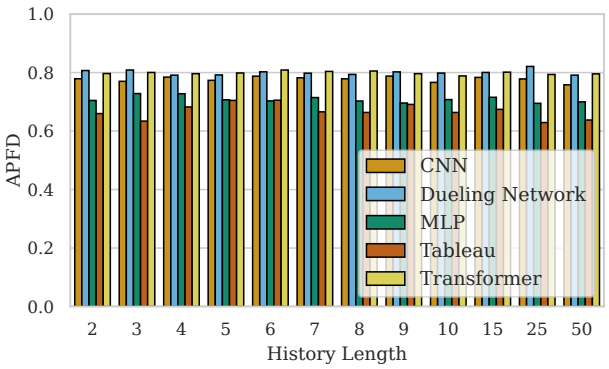

Figure 5: Impact of history length on model performance on the Paint Control dataset.

length. The MLP model's APFD score fluctuates more than any other model, peaking at lengths 2, 3, and 4. Initially, it achieves the highest APFD score with a history length of 2, but its performance steadily diminishes as the history length increases, eventually reaching parity with the Tableau agent. The CNN agent's performance remains stable until a history length of 25, after which it begins to decline. Both the Dueling Network and Transformer agents demonstrate consistent effectiveness in prioritizing test cases across varying history lengths.

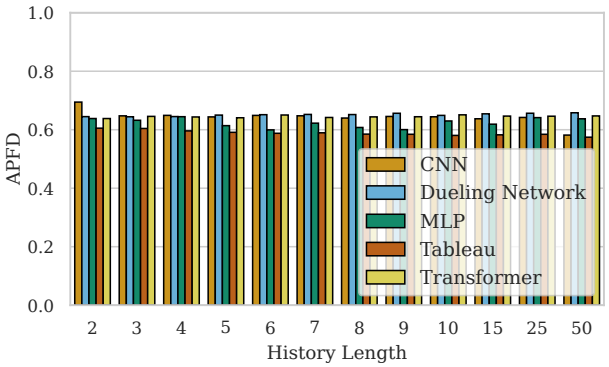

Figure 6: Impact of history length on model performance on the IOF/ROL dataset.

Further results from the Plejd dataset in Figure 7 reveal that the CNN agent's APFD scores remain stable, consistently ranking third among the models as history length increases. The Transformer model shows little variability, maintaining the second-highest scores. The Dueling network's APFD scores improve slightly with increased history lengths, indicating a benefit from processing longer sequences. Conversely, the MLP agent exhibits lower APFD scores with extended history lengths, underscoring the need for optimal tuning during model training. The Tableau agent, despite showing minor fluctuations, consistently underperforms compared to

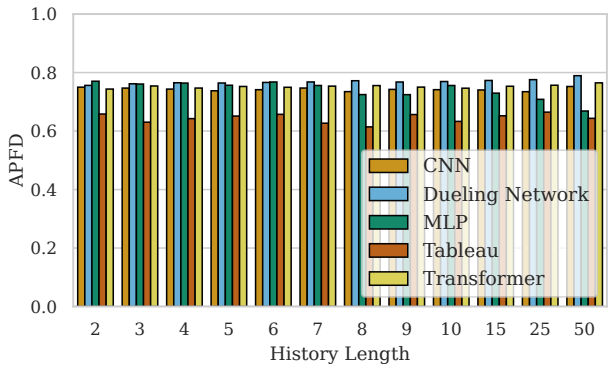

Figure 7: Impact of history length on model performance on the Plejd dataset.

other models.

5.2 RQ2: Comparison with RETECS Models

This section addresses **RQ2**: *How do the performance metrics of CNN, Dueling Network, and Transformer models compare to those of the RETECS models in TCP?* To answer this question, the NAPFD scores of the proposed models are compared with those of the RETECS models (MLP and Tableau) across the chosen datasets under varying time constraints using the NAPFD metric (Section 2.2.4). The reward function used for all subsequent results is Test Case Failure, introduced in Section 3.3.2 using Equation (5). Additionally, APFD scores for different reward functions are also considered. The results provide insights into the comparative performance of each model in prioritizing test cases effectively.

5.2.1 Effects of time limits

Figure 8 explores the scheduling time ratios and their impact on NAPFD scores across various models within the Paint Control dataset. The analysis indicates a uniform enhancement in NAPFD scores for all models as the scheduling time ratio increases, consistent with the expectation that extended scheduling periods allow for more optimal test case scheduling. The Dueling Network model demonstrates predominant performance throughout the time ratios, with the sole exception at a 50% ratio where the Transformer model marginally surpasses the other agents. Initially establishing and maintaining an early lead, the Dueling Network model exhibits robust efficiency across the tested intervals. The CNN model, initially the least performant, progressively improves, surpassing the Tableau and MLP agents at a 40% scheduling ratio. This initial underperformance could be attributed to the CNN's reliance on capturing spatial hierarchies, which may not be fully leveraged when scheduling time is severely limited. As the scheduling time ratio increases, the CNN has more opportunity to optimize the order of test cases by learning and utilizing spatial patterns more effectively. It is overtaken by the MLP agent for the two subsequent time ratios of 50 and 60 %, but gains the lead again for the remaining time ratios. The Transformer model consistently occupies the second rank, trailing the Dueling Network, except when it leads at the 50% ratio. The MLP model peaks at a 60% scheduling ratio, subsequently experiencing a slight decline in performance in the next time ratio, but it manages to improve slightly for the remaining schedueling times. Meanwhile, the Tableau model shows an early advantage over the CNN and MLP models between the 10% and 30% ratios but thereafter falls behind the performance of all the other models.

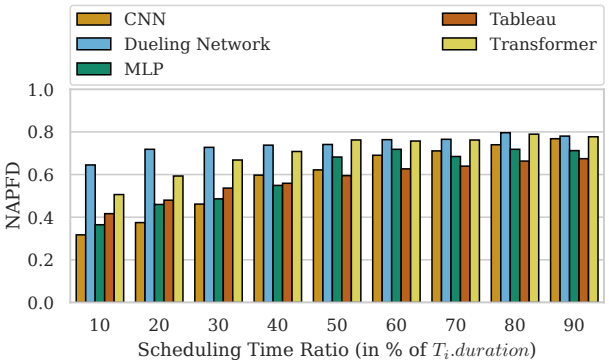

Figure 8: Performance under different time constraints on the Paint Control dataset.

Figure 9 presents the NAPFD scores for a range of machine learning models over various scheduling time ratios within the IOF/ROL dataset. Observations indicate that the CNN model generally parallels the Transformer model in performance, although it is surpassed at scheduling ratios of 10%, 30%, 80%, and 90% by a thin margin. The Dueling Network demonstrates robust performance across the scheduling time spectrum, with its scores only falling behind the CNN at a 70% scheduling ratio and the Transformer model at 30% and 80% ratios. The MLP model exhibits a progressive improvement in performance as the scheduling time ratio increases, yet consistently underperforms relative to the CNN, Dueling Network, and Transformer models. Its performance is comparable to the others at the 60% and 90% scheduling time ratios. Meanwhile, the Tableau model maintains higher NAPFD scores than the MLP up to a scheduling time ratio of 50%, beyond which it registers the lowest performance among the models evaluated.
Figure 10 illustrates the NAPFD scores for various models relative to the scheduling time ratios on the Plejd

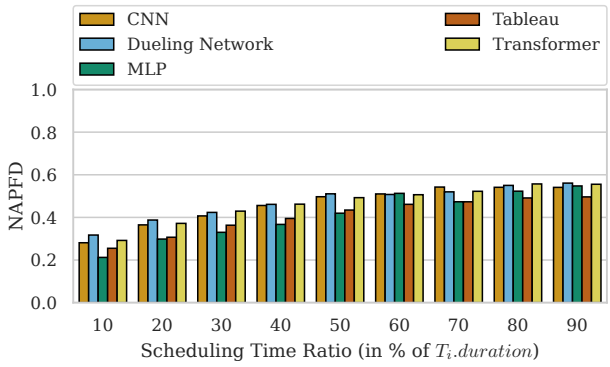

Figure 9: Performance under different time constraints on the IOF/ROL dataset.

dataset. The Dueling Network model demonstrates superior performance from the onset, achieving the highest NAPFD scores at the lowest time ratio and maintaining this lead across all subsequent ratios. In contrast, the Transformer model starts with lower NAPFD scores than the Dueling Network but shows a steady increase as the scheduling time ratio extends, aligning its scores with those of the CNN model at higher time ratios. Specifically, the CNN model is outperformed by the Transformer at time ratios of 10% and 40%; in the other instances, it exhibits better performance than the Transformer. The MLP model, initially ranking as the least effective, improves over time, ascending to third place at a scheduling time ratio of 50%. It also outperforms the tableau, CNN and Transformer in the following time ratio of 60%, later taking third place. Conversely, the Tableau model, which starts in second place to last, declines in performance compared to the other agents, dropping to last place at a time ratio of 50%.

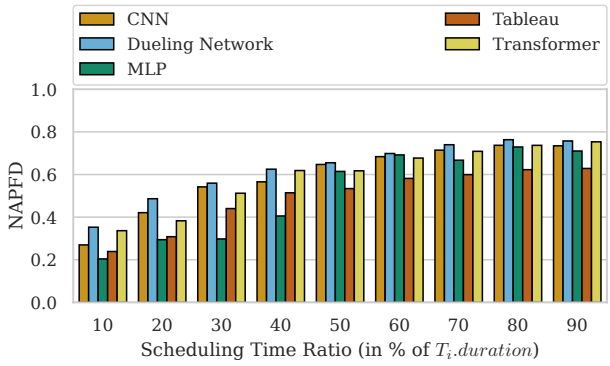

Figure 10: Performance under different time constraints on the Plejd dataset.

The effect of scheduling time ratios on NAPFD scores reveals consistent patterns and differences among various machine learning models across the Paint Control, IOF/ROL, and Plejd datasets. For the Paint Control dataset, the Dueling Network model consistently leads except at the 50% ratio, where the Transformer model takes the lead. The CNN model, initially the least performant, progressively improves, surpassing the Tableau model at 40% and the MLP model at 70%. The Transformer model generally remains in second place, trailing the Dueling Network, except at 70%. The MLP model peaks at 60% but declines thereafter, while the Tableau model falls behind after 20%.

In the IOF/ROL dataset, the Dueling Network leads, except at 30% and 80%, where the Transformer model surpasses it, and at 70% where the CNN model takes the lead. The CNN model performs similarly to the Transformer model, trailing slightly at 10%, 30%, 80%, and 90%. The MLP model improves progressively but remains behind the others, except at 60% and 90%. The Tableau model outperforms the MLP up to 50% but then registers the lowest scores.

Examining the Plejd dataset, the Dueling Network consistently leads from the outset. The Transformer model steadily improves, aligning with the CNN model at higher time ratios. The CNN model initially trails the Transformer at 10% and 40% but surpasses it at other ratios. The MLP model starts as the least effective but improves to third place at 50%, outperforming the Tableau, CNN, and Transformer models at 60%. The Tableau model starts second to last and declines to last place at 50%. Overall, the Dueling Network consistently performs well across the datasets, particularly at lower scheduling ratios. Its consistent dominance across all three datasets establishes it as the most effective agent in a variety of conditions. The Transformer model generally ranks second, showcasing steady improvement as the scheduling ratio increases. The CNN model improves at higher ratios across all datasets, initially underperforming but eventually surpassing other models in some cases. The MLP model shows varying performance, achieving its best results in the Plejd dataset but underperforming in the Paint Control and IOF/ROL datasets. The Tableau model typically underperforms relative to the others across all datasets, showing an early advantage over some models but generally declining in performance at higher scheduling ratios.

5.2.2 Impact of Reward Functions

The results presented in Fig. 11 and Table 3 illustrate the performance of the CNN, MLP, and Tableau agents on the three datasets using three different reward functions: Failcount, Tcfail, and Timerank.

Focusing on the Plejd dataset, with the Failcount reward, the CNN agent achieves an average APFD of 0.57 ± 0.18, slightly outperforming the MLP agent (0.56 ± 0.19) but trailing the Tableau agent (0.58 ± 0.18).

Despite this, the CNN agent shows more stability over the CI cycles, with less fluctuation compared to the MLP agent. This stability is evident in Fig. 11, where the CNN agent's APFD remains relatively steady above 0.5, indicating robust fault detection capabilities. Under the Tcfail reward, the CNN agent demonstrates superior performance, reaching a mean APFD of 0.74 ± 0.15, compared to 0.72 ± 0.17 for the MLP agent and 0.57 ± 0.18 for the Tableau agent. The CNN agent's advantage is clear, with a steady upward trend in APFD, surpassing 0.8 in the later CI cycles. In contrast, the MLP agent displays erratic performance, frequently falling below 0.6, while the Tableau agent remains relatively flat, hovering around an APFD of 0.55. For the Timerank reward, the CNN agent maintains a consistent performance, with a mean APFD of 0.58 ± 0.18. Although the Tableau agent performs better (0.66 ± 0.16), the CNN agent demonstrates more stability compared to the MLP agent (0.53 ± 0.21). Despite initial variability, the CNN agent eventually stabilizes above an APFD of 0.6 in Fig. 11. Overall, on the Plejd dataset, the results clearly indicate the superior performance of the CNN agent over both the MLP and Tableau agents. The CNN agent's ability to recognize intricate fault patterns through its convolutional layers results in a more effective fault detection strategy, consistently outperforming the MLP and Tableau agents across the Tcfail reward function and maintaining competitive performance under the other two reward functions.

Moving to the ABB Paint Control dataset, under the Failcount reward, the CNN agent achieves a mean APFD of 0.64 ± 0.22, which is comparable to the Tableau agent (0.64 ± 0.22). Despite this, Fig. 11 shows that the CNN agent displays less volatility compared to the MLP agent, which has a mean APFD of 0.52 ± 0.35. The CNN agent maintains a relatively stable APFD above 0.6. With the Tcfail reward, the CNN agent's performance improves significantly, achieving a mean APFD of 0.78 ± 0.24. This places the CNN agent well ahead of the Tableau agent (0.63 ± 0.23). The trend lines in Fig. 11 highlight the CNN agent's steady increase in APFD, particularly in the later CI cycles. For the Timerank reward, the CNN agent delivers a stable performance, reaching a mean APFD of 0.65 ± 0.23. The CNN agent shows consistent results, outperforming the Tableau agent (0.66 ± 0.21) and MLP agent (0.55 ± 0.33). The CNN agent's APFD stabilizes above 0.6 across the CI cycles.

In the ABB IOF/ROL dataset, with the Failcount reward, the CNN agent achieves a mean APFD of 0.58 ± 0.18, closely matching the Tableau agent (0.58 ± 0.18). Fig. 11 shows that the CNN agent maintains a steady APFD above 0.5, indicating stable fault detection performance. Under the Tcfail reward, the CNN agent achieves a mean APFD of 0.64 ± 0.19, outperforming the Tableau agent, which only manages a mean APFD of 0.58 ± 0.18. The trend lines in Fig. 11 reveal a gradual upward trajectory for the CNN agent, consistently improving its APFD across CI cycles. For the Timerank reward, the CNN agent's performance remains steady, reaching a mean APFD of 0.58 ± 0.18. Despite matching the Tableau agent (0.58 ± 0.18), the CNN agent maintains stable performance over the CI cycles, with an APFD above 0.5 throughout.

The results presented in Fig. 12 and Table 3 illustrate the performance of the Dueling Network agent compared to the MLP and Tableau agents on the three distinct datasets using the three reward functions.

Analyzing the Plejd dataset, under the Failcount reward, the Dueling Network agent achieves a mean APFD of 0.59 ± 0.19, surpassing both the MLP agent (0.56 ± 0.19) and the Tableau agent (0.58 ± 0.18). Fig. 12 shows the Dueling Network agent maintaining a relatively steady APFD above 0.55 across the CI cycles, indicating robust performance. With the Tcfail reward, the Dueling Network agent exhibits the best performance among the agents, achieving a mean APFD of 0.78 ± 0.15, compared to the MLP agent (0.72 ± 0.17) and the Tableau agent (0.57 ± 0.18). The agent's APFD consistently exceeds 0.8 in later CI cycles. For the Timerank reward, the Dueling Network agent maintains a competitive mean APFD of 0.63 ± 0.18, outperforming the MLP agent (0.53 ± 0.21) and closely following the Tableau agent (0.66 ± 0.16). Despite initial fluctuations, the Dueling Network agent eventually stabilizes above an APFD of 0.6.

Looking at the ABB Paint Control dataset, under the Failcount reward, the Dueling Network agent achieves a mean APFD of 0.78 ± 0.24, significantly outperforming the MLP agent (0.52 ± 0.35) and the Tableau agent (0.64 ± 0.22). Fig. 12 reveals that the Dueling Network agent's APFD remains relatively stable, surpassing 0.7 across the CI cycles. When using the Tcfail reward, the Dueling Network agent maintains the highest mean APFD of 0.80 ± 0.23, ahead of the Tableau agent (0.63 ± 0.23). The Dueling Network agent's APFD consistently trends upwards, eventually stabilizing above 0.8. For the Timerank reward, the Dueling Network agent continues to deliver strong performance, reaching a mean APFD of 0.78 ± 0.24. Despite fluctuations in the early CI cycles, the Dueling Network agent stabilizes above an APFD of 0.75, outperforming both the MLP agent (0.55 ± 0.33) and the Tableau agent (0.66 ± 0.21).

Examining the ABB IOF/ROL dataset, under the Failcount reward, the Dueling Network agent achieves a mean APFD of 0.60 ± 0.21, narrowly surpassing the MLP agent (0.60 ± 0.20) and the Tableau agent (0.58 ± 0.18). Fig. 12 shows the Dueling Network agent's APFD remaining relatively steady above 0.55. With the Tcfail reward, the Dueling Network agent reaches a mean APFD of 0.65 ± 0.20, slightly ahead of the MLP agent (0.64 ± 0.19) and on par with the

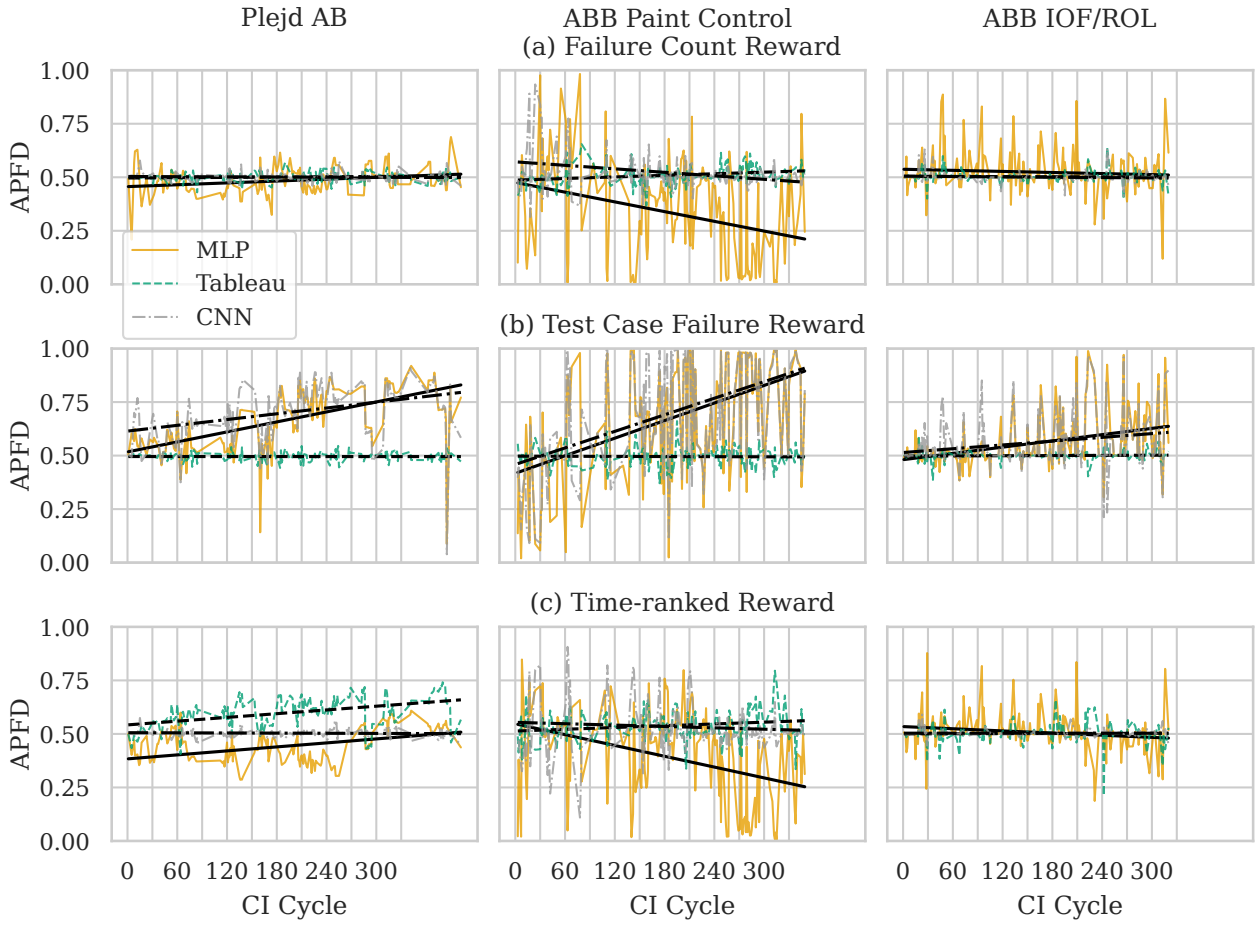

Figure 11: Performance of the CNN model using the different reward functions (4), (5) and (6) on the datasets.

Tableau agent (0.58 ± 0.18). The Dueling Network agent's APFD consistently trends upwards, eventually stabilizing around 0.65. For the Timerank reward, the Dueling Network agent maintains a competitive mean APFD of 0.60 ± 0.21, outperforming the MLP agent (0.59 ± 0.20) and the Tableau agent (0.58 ± 0.18). The Dueling Network agent exhibits relatively stable performance, with its APFD remaining above 0.55 across the CI cycles.

The results presented in Fig. 13 and Table 3 illustrate the performance of the Transformer, MLP, and Tableau agents on the three datasets and rewardfunction.

For the Plejd dataset, using the Failcount reward, the Transformer agent achieves an average APFD of 0.57 ± 0.18, slightly outperforming the MLP agent (0.56 ± 0.19) but trailing the Tableau agent (0.58 ± 0.18). Fig. 13 shows that the Transformer agent maintains a relatively steady APFD above 0.5, indicating stable fault detection performance throughout the CI cycles. When evaluated using the Tcfail reward, the Transformer agent demonstrates superior performance, reaching a mean APFD of 0.75 ± 0.15, compared to 0.72 ± 0.17 for the MLP agent and 0.57 ± 0.18 for the Tableau agent. The Transformer's upward trend in APFD, surpassing 0.8 in the later CI cycles, is evident in Fig. 13. In contrast, the MLP agent shows erratic performance, frequently falling below 0.6, while the Tableau agent remains relatively flat. For the Timerank reward, the Transformer agent maintains consistent performance, with a mean APFD of 0.57 ± 0.18, outperforming the MLP agent (0.53 ± 0.21) but trailing the Tableau agent (0.66 ± 0.16). Despite initial variability, the Transformer agent eventually stabilizes above an APFD of 0.6 in Fig. 13.

In the ABB Paint Control dataset, with the Failcount reward, the Transformer agent achieves a mean APFD of 0.63 ± 0.22, which is comparable to the Tableau agent (0.64 ± 0.22). However, Fig. 13 shows that the Transformer agent displays less volatility compared to the MLP agent, which has a mean APFD of 0.52 ± 0.35. With the Tcfail reward, the Transformer agent's performance improves significantly, achieving a mean APFD of 0.80 ± 0.22, outpacing the Tableau agent (0.63 ± 0.23). The trend lines in Fig. 13 highlight the steady increase

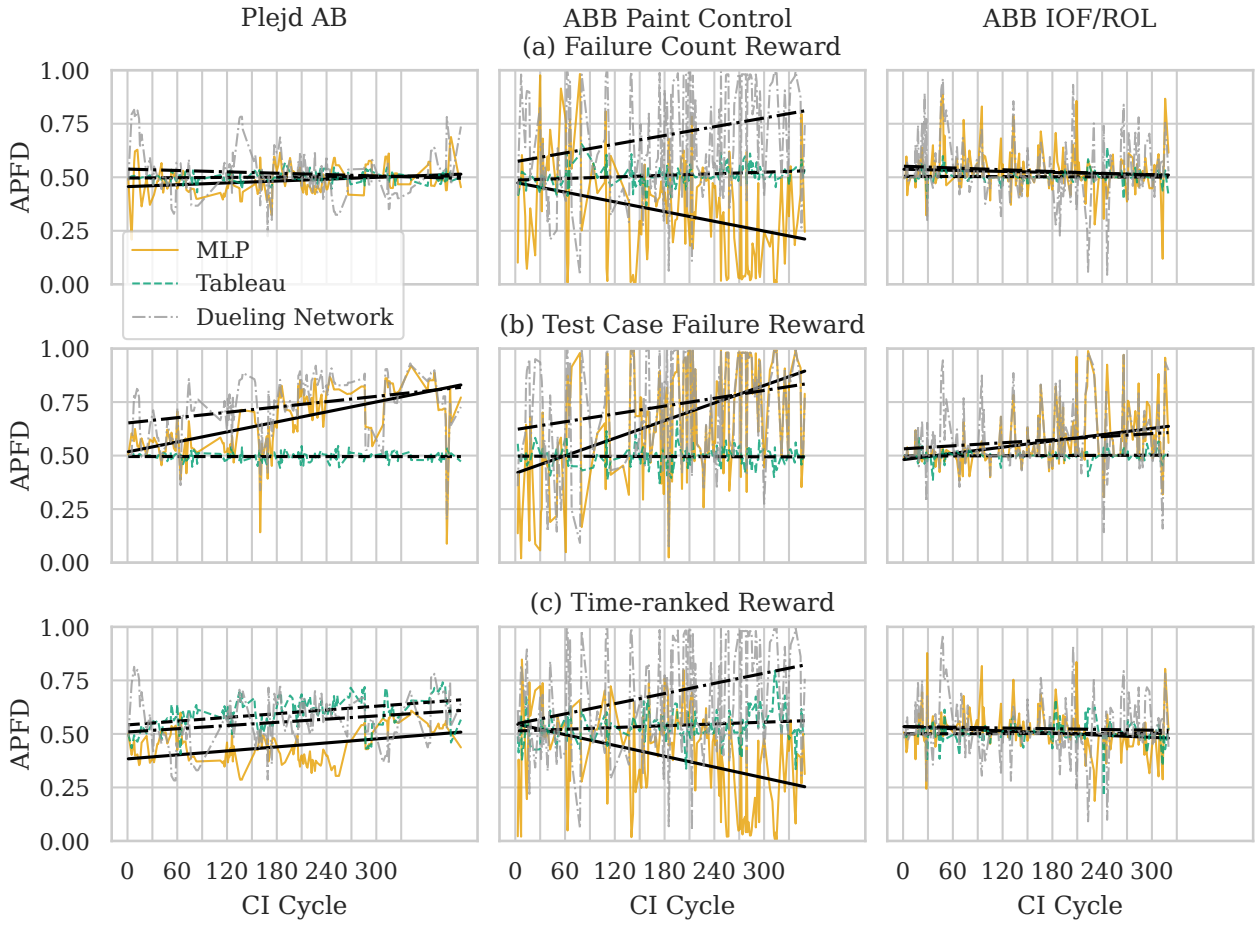

Figure 12: Performance of the Dueling Network agent using different reward functions (Equations 4, 5, and 6) on the datasets. The MLP and Tableau data correspond to the same data presented in FIGs 11 and 13.

in APFD for the Transformer agent, particularly in the later CI cycles. For the Timerank reward, the Transformer agent delivers a stable performance, reaching a mean APFD of 0.64 ± 0.22. While this is slightly lower than the Tableau agent (0.66 ± 0.21), the Transformer agent shows consistent results, outperforming the MLP agent (0.55 ± 0.33). The Transformer agent's APFD stabilizes above 0.6 across the CI cycles.

When considering the ABB IOF/ROL dataset, with the Failcount reward, the Transformer agent achieves a mean APFD of 0.58 ± 0.18, matching the performance of the Tableau agent (0.58 ± 0.18). Fig. 13 shows that the Transformer agent maintains a steady APFD above 0.5, indicating stable fault detection performance. Under the Tcfail reward, the Transformer agent achieves a mean APFD of 0.64 ± 0.19, outperforming the Tableau agent, which only manages a mean APFD of 0.58 ± 0.18. The trend lines in Fig. 13 reveal a gradual upward trajectory for the Transformer agent, consistently improving its APFD across CI cycles. For the Timerank reward, the Transformer agent's performance remains steady, reaching a mean APFD of 0.58 ± 0.18. Despite matching the Tableau agent (0.58 ± 0.18), the Transformer agent maintains stable performance over the CI cycles, with an APFD above 0.5 throughout.

The results presented in Fig. 14 and Table 3 illustrate the performance of the CNN, Dueling Network, and Transformer agents across the three datasets, using the three reward.

Looking at the Plejd dataset, under the Failcount reward, the Dueling Network agent achieves the highest mean APFD of 0.59 ± 0.19, followed closely by the Tableau agent (0.58 ± 0.18), CNN agent (0.57 ± 0.18), and Transformer agent (0.57 ± 0.18). The MLP agent falls behind with a mean APFD of 0.56 ± 0.19. Despite these marginal differences, the Dueling Network agent shows a consistent upward trend over the CI cycles in Fig. 14, indicating stable fault detection. Switching to the Tcfail reward, the Dueling Network agent leads with a mean APFD of 0.78 ± 0.15, followed by the Transformer agent (0.75 ± 0.15), CNN agent (0.74 ± 0.15), MLP agent (0.72 ± 0.17), and finally the Tableau agent

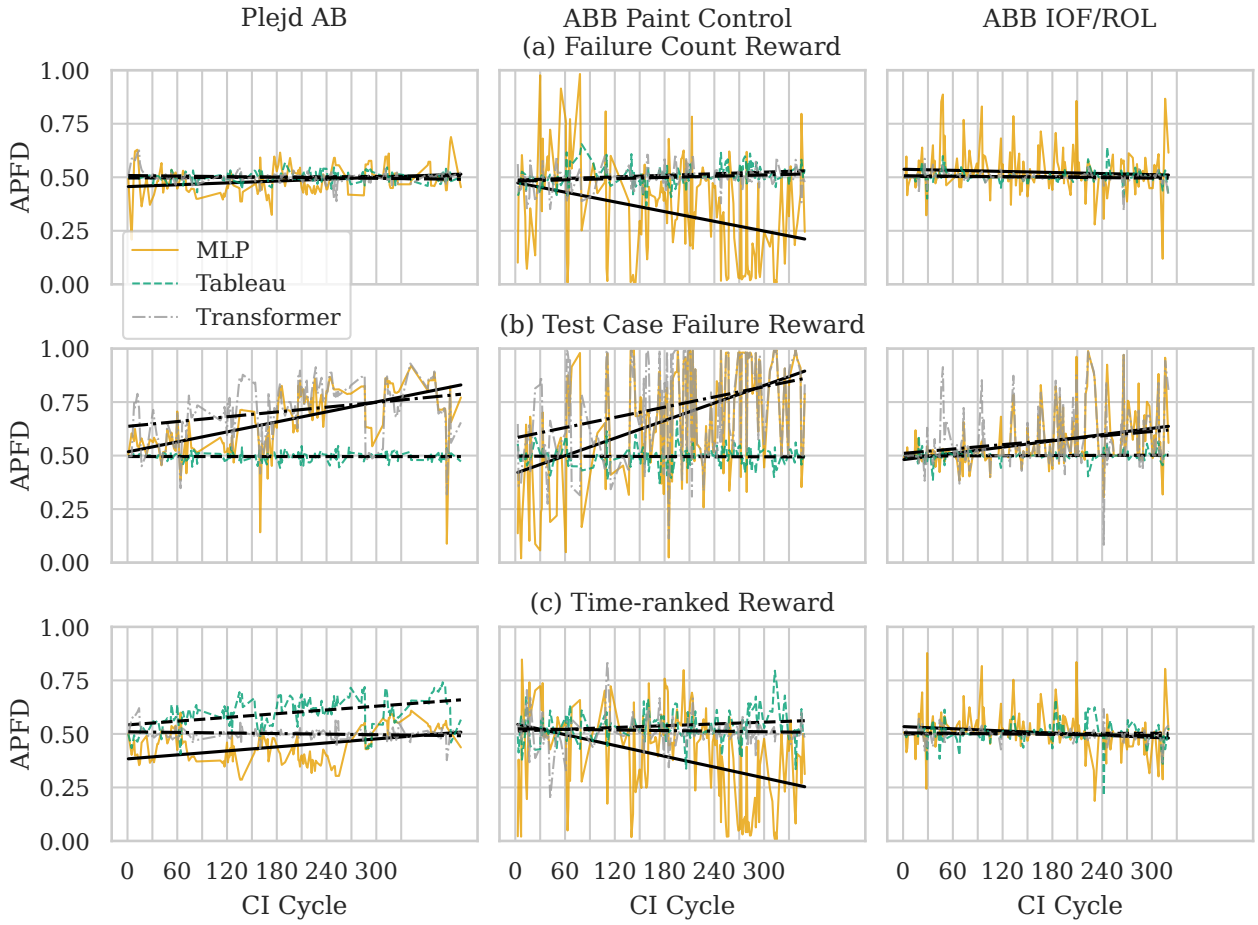

Figure 13: Performance of the Dueling Network agent using different reward functions (Equations 4, 5, and 6) on the datasets. The MLP and Tableau data correspond to the same data presented in FIGs 11 and 12.

(0.57 ± 0.18). Fig. 14 reveals that the Dueling Network agent's APFD consistently exceeds 0.8 in the later CI cycles, while the CNN and Transformer agents maintain stable performance above 0.7. For the Timerank reward, the Tableau agent achieves the highest mean APFD of 0.66 ± 0.16, followed by the Dueling Network agent (0.63 ± 0.18), CNN agent (0.58 ± 0.18), Transformer agent (0.57 ± 0.18), and MLP agent (0.53 ± 0.21). Despite initial fluctuations, the Dueling Network agent stabilizes above an APFD of 0.6, while the CNN and Transformer agents demonstrate relatively steady performance.

Analyzing the ABB Paint Control dataset, under the Failcount reward, the Dueling Network agent delivers the best performance, achieving a mean APFD of 0.78 ± 0.24. The CNN and Tableau agents follow closely with matching APFDs (0.64 ± 0.22), while the Transformer agent achieves a comparable 0.63 ± 0.22. The MLP agent trails behind with 0.52 ± 0.35. Fig. 14 shows the Dueling Network agent maintaining a steady APFD above 0.7. When using the Tcfail reward, the Dueling Network and Transformer agents both achieve the highest mean APFDs of 0.80 ± 0.23 and 0.80 ± 0.22, respectively. The CNN agent is not far behind with an APFD of 0.78 ± 0.24, while the Tableau agent lags behind at 0.63 ± 0.23. The MLP agent achieves a mean APFD of 0.77 ± 0.26. The trend lines in Fig. 14 show that all three leading agents maintain stable performance above 0.75. For the Timerank reward, the Dueling Network agent remains at the top, achieving an APFD of 0.78 ± 0.24. The CNN agent follows closely with an APFD of 0.65 ± 0.23, while the Transformer agent achieves a comparable 0.64 ± 0.22. The Tableau agent reaches an APFD of 0.66 ± 0.21, while the MLP agent remains behind with 0.55 ± 0.33.

Focusing on the ABB IOF/ROL dataset, under the Failcount reward, the Dueling Network agent achieves the highest mean APFD of 0.60 ± 0.21, followed by the MLP agent (0.60 ± 0.20), CNN agent (0.58 ± 0.18), Transformer agent (0.58 ± 0.18), and Tableau agent (0.58 ± 0.18). The Dueling Network agent maintains a steady APFD above 0.55, as seen in Fig. 14. Switching

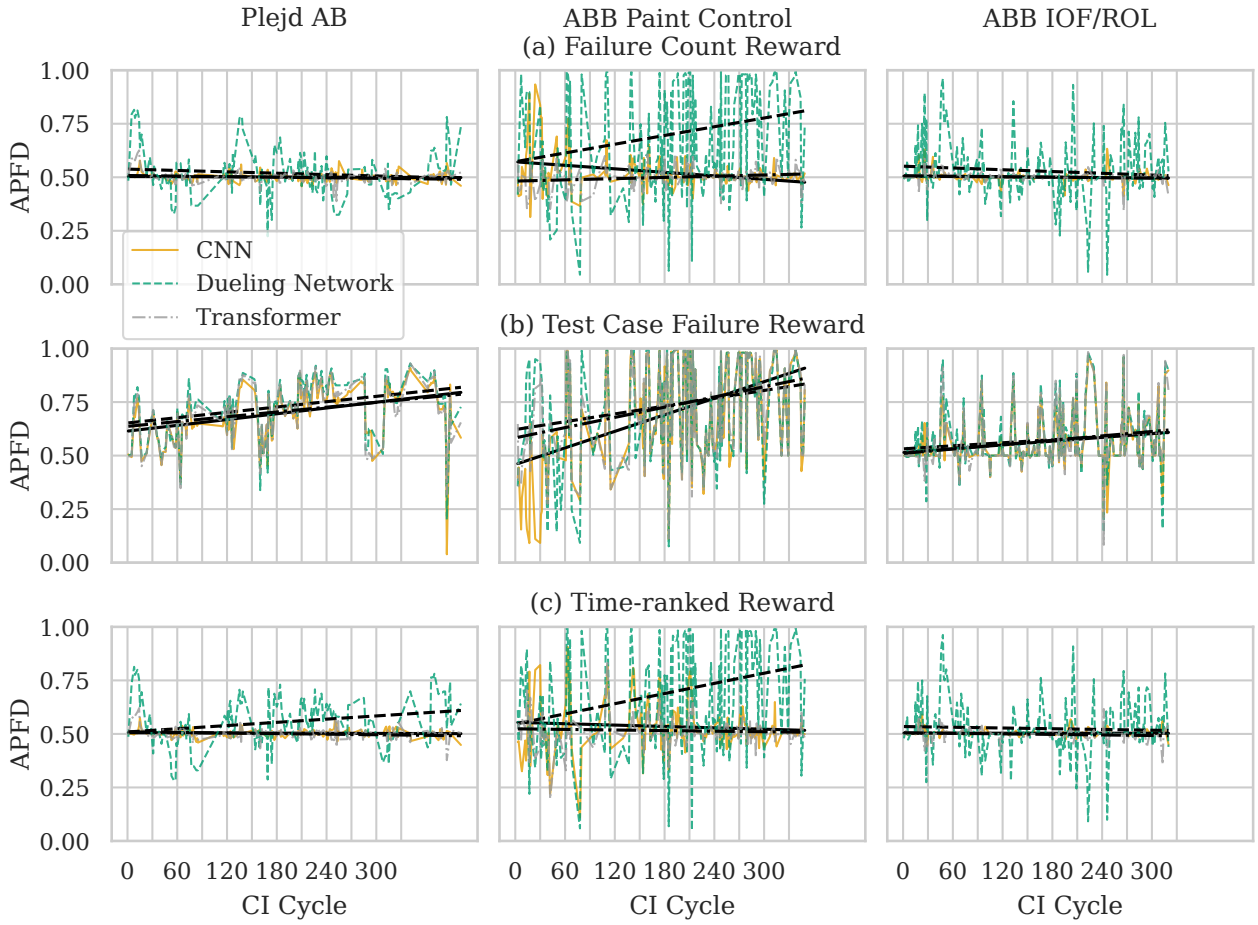

Figure 14: Performance of the three architectures using different reward functions (Equations 4, 5, and 6) on the datasets. The data is the same as presented in FIGs 11, 12, and 13.

to the Tcfail reward, the Dueling Network agent reaches the highest mean APFD of 0.65 ± 0.20, while the Transformer and CNN agents both achieve 0.64 ± 0.19. The MLP agent trails with 0.64 ± 0.19, while the Tableau agent achieves 0.58 ± 0.18. Fig. 14 shows that the Dueling Network agent maintains a steady upward trend, stabilizing around 0.65. For the Timerank reward, the Dueling Network agent achieves a mean APFD of 0.60 ± 0.21, narrowly outperforming the CNN agent (0.58 ± 0.18), Transformer agent (0.58 ± 0.18), and Tableau agent (0.58 ± 0.18). The MLP agent remains close with an APFD of 0.59 ± 0.20. Despite fluctuations, the Dueling Network agent exhibits relatively stable performance, with its APFD remaining above 0.55 across the CI cycles.

Overall, across the three datasets, the Dueling Network agent consistently demonstrates superior performance, particularly excelling under the Tcfail reward function. The CNN and Transformer agents follow closely, each showing strengths in specific areas. The CNN agent's ability to recognize intricate fault patterns through its convolutional layers results in an effective fault detection strategy, especially in the Plejd dataset. The Transformer agent remains a robust and reliable choice for test case prioritization, effectively recognizing complex fault patterns and delivering stable performance across the datasets. Although the Dueling Network agent occasionally surpasses both the CNN and Transformer agents, particularly in the ABB Paint Control dataset, all three models consistently outperform the MLP and Tableau agents in most scenarios.

Dataset	Agent	Reward Function	$\mu \pm \sigma$
	MLP	Failcount	0.56 ± 0.19
	Tableau	Failcount	0.58 ± 0.18
	CNN	Failcount	0.57 ± 0.18
	Duel	Failcount	0.59 ± 0.19
	Transformer	Failcount	0.57 ± 0.18
	MLP	Tcfail	0.72 ± 0.17
	Tableau	Tcfail	0.57 ± 0.18
Plejd	CNN	Tcfail	0.74 ± 0.15
	Duel	**Tcfail**	**0.78 ± 0.15**
	Transformer	Tcfail	0.75 ± 0.15
	MLP	Timerank	0.53 ± 0.21
	Tableau	Timerank	0.66 ± 0.16
	CNN	Timerank	0.58 ± 0.18
	Duel	Timerank	0.63 ± 0.18
	Transformer	Timerank	0.57 ± 0.18
	MLP	Failcount	0.52 ± 0.35
	Tableau	Failcount	0.64 ± 0.22
	CNN	Failcount	0.64 ± 0.22
	Duel	Failcount	0.78 ± 0.24
	Transformer	Failcount	0.63 ± 0.22
	MLP	Tcfail	0.77 ± 0.26
	Tableau	Tcfail	0.63 ± 0.23
Paint Control	CNN	Tcfail	0.78 ± 0.24
	Duel	**Tcfail**	**0.80 ± 0.23**
	Transformer	**Tcfail**	**0.80 ± 0.22**
	MLP	Timerank	0.55 ± 0.33
	Tableau	Timerank	0.66 ± 0.21
	CNN	Timerank	0.65 ± 0.23
	Duel	Timerank	0.78 ± 0.24
	Transformer	Timerank	0.64 ± 0.22
	MLP	Failcount	0.60 ± 0.20
	Tableau	Failcount	0.58 ± 0.18
	CNN	Failcount	0.58 ± 0.18
	Duel	Failcount	0.60 ± 0.21
	Transformer	Failcount	0.58 ± 0.18
	MLP	Tcfail	0.64 ± 0.19
	Tableau	Tcfail	0.58 ± 0.18
IOF/Rol	CNN	Tcfail	0.64 ± 0.19
	Duel	**Tcfail**	**0.65 ± 0.20**
	Transformer	Tcfail	0.64 ± 0.19
	MLP	Timerank	0.59 ± 0.20
	Tableau	Timerank	0.58 ± 0.18
	CNN	Timerank	0.58 ± 0.18
	Duel	Timerank	0.60 ± 0.21
	Transformer	Timerank	0.58 ± 0.18

Table 3: Summary of mean APFD score and standard deviation for different agents and reward functions across the datasets from Figures 11, 12 and 13 over 20 iterations.

5.3 RQ3: Comparison with Heuristics and Contemporary Models

This subsection addresses **RQ3**: How do the proposed neural network models compare with heuristic and contemporary models in enhancing test case prioritization within CI environments? To answer this question, the proposed CNN, Dueling Network, and Transformer models are compared with heuristic approaches, including Sorting, Weighting, and Random, across multiple datasets. In the Sorting method, test cases are prioritized based on their recent execution results, giving higher priority to those that failed recently. The Weighting method assigns priorities based on a weighted sum of test case features such as duration and failure history. The Random method prioritizes test cases in a random order, serving as a baseline. The comparison provides insights into the effectiveness of each model in prioritizing test cases within continuous integration environments. Test Case Failure was selected as the reward function for all subsequent tests due to its performance in previous tests.

5.3.1 Comparison of Models with Heuristic Approaches

The results for the Plejd dataset, as shown in the first plot of Figure 15, reveal a consistent trend across CI cycles. The CNN agent generally outperforms the heuristic approaches, as evidenced by the predominantly negative APFD differences. The Sorting heuristic underperforms compared to the CNN agent, showing negative APFD differences throughout all CI cycles. The CNN agent achieves its largest advantage over the Sorting heuristic in CI cycles 210-240, with a difference of just over 0.25 in APFD. However, both the Sorting and Weighting heuristics slightly outperform the CNN agent in the first 30 CI cycles, achieving positive APFD differences of around 0.1. After CI cycle 30, both heuristics are consistently outperformed by the CNN agent in subsequent CI cycles, with the highest gap for both in CI cycle 330, where the CNN agent achieves an APFD lead of slightly over 0.25. The Random heuristic consistently underperforms the CNN agent across all CI cycles. The gap narrows in the final cycle, where the heuristics perform very similarly to the CNN agent, this drop in performance can be seen in Fig. 14 (b) for the Plejd dataset, where the CNN doesn't recover as well as the other models (Dueling Network and Transformer), leading to a comparative performance with the heuristics. The results for the Plejd dataset highlight the consistent effectiveness of the CNN agent in outperforming all three heuristic approaches, particularly in later CI cycles (180 to 360).

In the ABB Paint Control dataset, the CNN model initially exhibits negative APFD differences compared to the Sorting, Weighting, and Random heuristics, as seen in the first 60 CI cycles. During this period, the CNN model is significantly outperformed by both the Sorting and Weighting heuristics, which achieve substantial performance leads, with both the Weighting and Sorting heuristics leading by over 0.5 in APFD in the first 30 CI cycles. This lead diminishes around the 60th CI cycle, where the CNN model performs on par with the Sorting and Weighting heuristics. It also begins to consistently outperform the Random heuristic from that point onward, obtaining the biggest difference between the 270th and 310th CI cycles, with an advantage of approximately 0.3 in APFD. The Sorting and Weighting methods start to oscillate in performance compared to the CNN model, with a peak noted between CI cycles 180 and 210. The CNN model then outperforms all the heuristics between CI cycles 240 and 270 but is overtaken in APFD scores for the remaining cycles. The CNN model shows a strong ability to adapt and improve over time, achieving substantial gains in APFD scores and outperforming the heuristics during certain intervals.

Analyzing the ABB IOF/ROL dataset, the performance differences between the CNN model and the heuristic approaches are extremely narrow, with minimal deviations observed throughout the CI cycles. The CNN model consistently outperforms the Random heuristic but is marginally outperformed by the Sorting and Weighting heuristics in almost all CI cycles, particularly between 0-60 and 240-300. The CNN model achieves its largest advantage over the Random heuristic between CI cycles 90-120 and 210-240, where it leads by approximately 0.1 in APFD.

Figure 16 shows the Dueling Network model's performance against the heuristic methods. In the Plejd AB dataset, the Sorting and Weighting heuristics consistently underperform compared to the dueling model, showing negative APFD differences throughout all CI cycles except for the first 30 cycles, where they achieve a positive APFD score compared the Dueling Network agent. In CI cycles 330-360, the Dueling model achieves its largest advantage over the Sorting heuristic, with a difference of approximately 0.3 in APFD score. Similarly, the Weighting heuristic struggles to match the dueling model's performance, with the worst performing cycle in the same interval, where it reaches a APFD difference of around 0.3. The Random heuristic demonstrates the poorest performance, with substantial negative differences across all CI cycles. In CI cycle 210, the dueling model achieves its most significant advantage over the Random heuristic, with APFD differences of almost 0.4.

Considering the ABB Paint Control dataset, the Dueling Network model exhibits mixed performance relative to the Sorting and Weighting heuristics. In the initial 30 CI cycles, the model is notably outperformed by both heuristics, with the Sorting heuristic achieving a

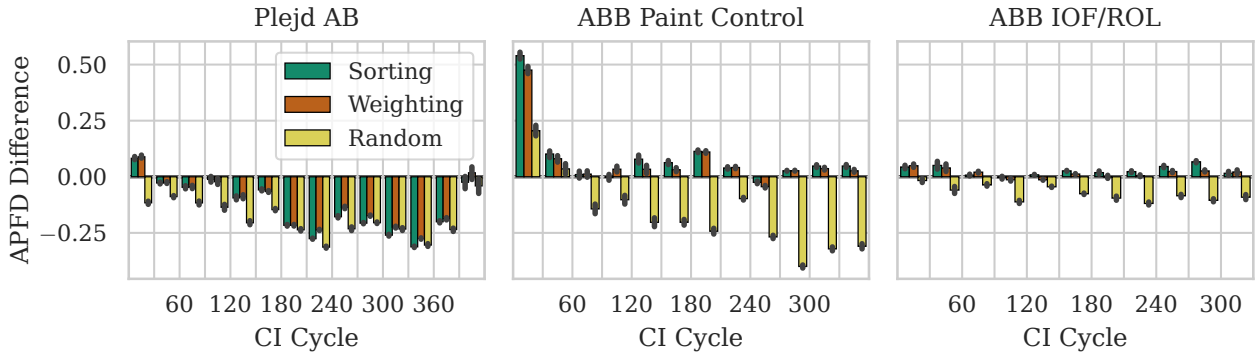

Figure 15: APFD performance difference for CNN agent compared to heuristic methods. Each group of bars compares 30 CI cycles.

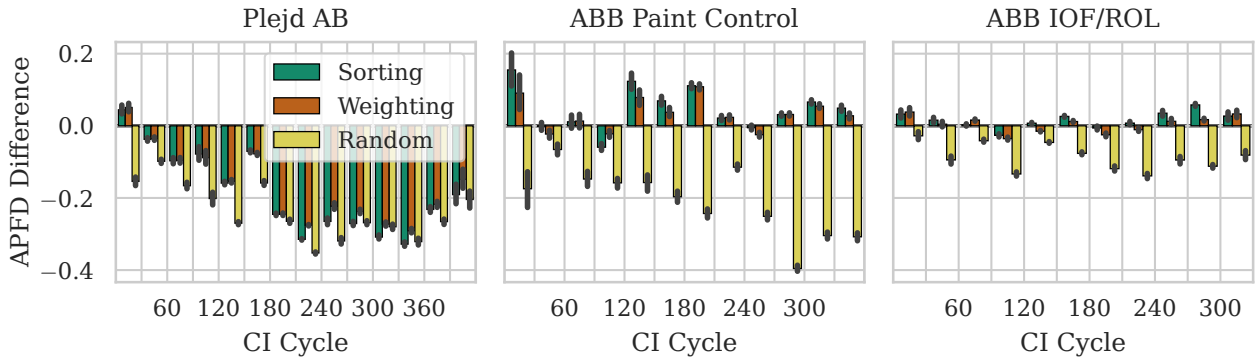

Figure 16: APFD performance difference for Dueling Network agent compared to heuristic methods. Each group of bars compares 30 CI cycles.

greater APFD difference of over 0.1, and the Weighting heuristic leading by approximately 0.05. Conversely, the model consistently outperforms the Random heuristic throughout all CI cycles, achieving its maximum advantage between cycles 270 and 300, with an APFD difference of around 0.4. From CI cycles 30 to 120, the performance of the dueling network model aligns more closely with that of the Sorting and Weighting heuristics, even surpassing both within the interval from cycles 90 to 120. However, beyond cycle 120, both heuristic methods regain superiority over the dueling network model, except between cycles 240 and 270. During these cycles, the model's APFD scores are comparable to those of the Sorting heuristic, falling within the standard deviation, and are marginally lower than those of the Weighting heuristic.

Regarding the ABB IOF/ROL dataset, the dueling network model outperforms the Random heuristic across all CI cycles but shows mixed performance compared to the Sorting and Weighting heuristics. In the initial 30 CI cycles, the dueling network model is marginally outperformed by the Sorting and Weighting heuristics, with positive APFD differences of around 0.04 in favor of both heuristics. From CI cycles 30 to 60, the dueling network model continues to trail behind the Sorting and Weighting heuristics, though the gap narrows as it begins to achieve scores within 0.03 of both heuristics. The dueling network model reaches near-parity with the Sorting and Weighting heuristics from CI cycles 90 to 240, sometimes achieving a greater APFD score. However, from CI cycle 240 onwards, both heuristics regain their lead, consistently outperforming the dueling network model up to the final CI cycle. Despite these differences, the dueling network model maintains a consistent advantage over the Random heuristic, with the largest differences observed between CI cycles 210 and 240, where the dueling network model leads by approximately 0.15 in APFD.

Figure 17 illustrates the Transformer models performance compared to the three heuristic methods. Analyzing the Plejd dataset, the Sorting and Weighting heuristics outperform the Transformer model in the first 30 CI cycles, with approximate APFD advantages of 0.08 and 0.09, respectively. However, from the second CI cycle (30-60) onwards, the Transformer model consistently surpasses both heuristics, gradually widening

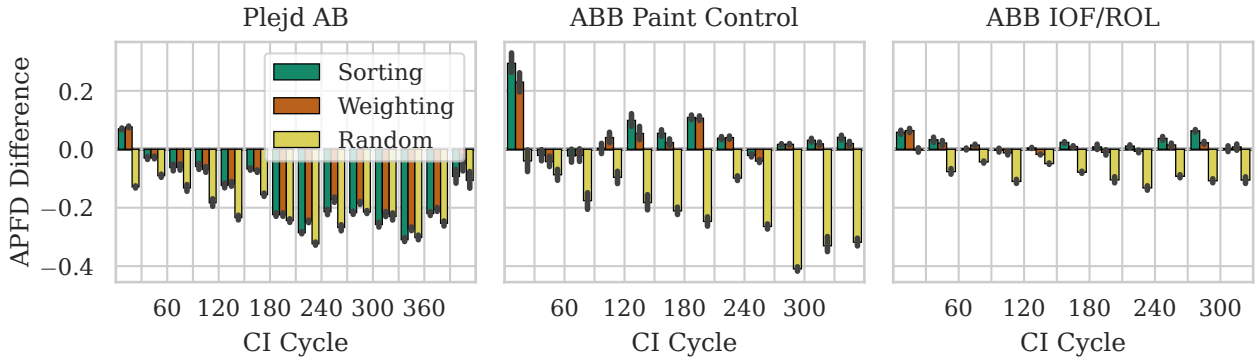

Figure 17: APFD performance difference for Transformer agent compared to heuristic methods. Each group of bars compares 30 CI cycles.

its advantage. The gap reaches its peak at CI cycle 330-360, where the Transformer model holds a lead of 0.28 over the Sorting heuristic and 0.25 over the Weighting heuristic. Despite a slight reduction in the advantage towards the end of the CI cycles, the Transformer model finishes with an approximate APFD advantage of 0.08 over the Sorting heuristic and 0.05 over the Weighting heuristic. In comparison to the Random heuristic, the Transformer model consistently performs better across all CI cycles. In the first 30 cycles, the Transformer model already leads with an approximate APFD advantage of 0.16, and this lead widens progressively, reaching a peak of 0.3 in the 240th CI cycle. Despite minor fluctuations, the Transformer model maintains a substantial lead over the Random heuristic throughout, ending with an approximate advantage of 0.1.

For the ABB Paint Control dataset the Sorting and Weighting heuristics initially outperform the Transformer model in the first 30 CI cycles, with approximate APFD advantages of 0.3 and 0.24, respectively. However, the Transformer model quickly closes the gap and begins to outperform both heuristics from CI cycles 30 to 60, with marginal advantages of 0.02 and 0.04 over the Sorting and Weighting heuristics, respectively. From CI cycles 60 to 120, the Transformer model maintains comparable performance with the Sorting and Weighting heuristics, with APFD differences fluctuating around 0. In the subsequent CI cycles 120 to 210, the Sorting and Weighting heuristics regain their lead over the Transformer model, achieving peak advantages of apart form the first cycles of 0.1. However, the Transformer model narrows the gap again, achieving near-parity with the heuristics from CI cycles 240 to 330. Compared to the Random heuristic, the Transformer model consistently demonstrates superior performance across all CI cycles. In the first 30 cycles, the Transformer model already leads with an approximate APFD advantage of 0.03, and this lead widens progressively to a peak of 0.31 in CI cycle 300. Despite minor fluctuations, the Transformer model maintains a substantial lead over the Random heuristic throughout, ending with an approximate advantage of 0.26.

Comparing the ABB IOF/ROL dataset, the Sorting heuristic outperforms the Transformer model in the first 30 CI cycles with an approximate APFD advantage of 0.06. However, this advantage narrows to 0.03 in the next 30 cycles and eventually levels off, with both approaches achieving comparable performance between CI cycles 60 and 150. From CI cycle 150 onwards, the Transformer model narrows the gap further, with APFD differences fluctuating between 0.01 and 0.06 in favor of the Sorting heuristic. The Weighting heuristic also outperforms the Transformer model in the first 30 CI cycles with an approximate APFD advantage of 0.07. However, this advantage diminishes quickly, and the Transformer model matches the Weighting heuristic's performance over the next 30 CI cycles. From CI cycles 90 to 120, the Transformer model gains a slight advantage, leading by 0.01 to 0.02, but the Weighting heuristic regains parity in subsequent CI cycles, with APFD differences ranging from 0.01 to 0.02. Compared to the Random heuristic, the Transformer model consistently demonstrates superior performance across all CI cycles. Although both models initially achieve similar APFD scores, the Transformer model progressively widens its advantage from CI cycles 30 to 90, leading by up to 0.11. This lead increases further in subsequent cycles, reaching a peak advantage of 0.16 between CI cycles 210 and 240. Despite minor fluctuations, the Transformer model maintains a significant lead over the Random heuristic throughout, concluding with an approximate APFD advantage of 0.1.

5.3.2 Comparison of Neural Network Models with DeepOrder and Bayesian Models

This subsection addresses **RQ3**: *How do the proposed neural network models compare with heuristic and contemporary models in enhancing test case prioritization within CI environments?* To answer this question, the performance of the CNN, Dueling Network, and Transformer models developed in this thesis is compared with that of the DeepOrder approach, as documented by Sharif et al. [13], and the Bayesian modeling method used in an associated study by the same company that supported this research [12].

A. DeepOrder Method The DeepOrder model uses a MLP for test case prioritization. It leverages deep learning techniques to rank test cases based on historical execution data. The input features for the DeepOrder model include execution status from multiple CI cycles, test case duration, the time of the last execution, the absolute difference in execution status between the most recent and least recent CI cycles, and the number of times a test case's execution status has changed from pass to fail. These features allow the model to make informed prioritization decisions based on the test case history and performance characteristics [13], thereby incorporating more features than the models developed in this thesis.

B. DeepOrder Results The following results are taken directly from the paper by Sharif et al. [13]:

- **Paint Control dataset**: APFD of 0.76 and NAPFD of 0.52
- **IOF/ROL dataset**: APFD of 0.67 and NAPFD of 0.56

C. Bayesian Modeling Method The Bayesian modeling approach, tailored to the needs of small to medium-sized enterprises, evaluates test case prioritization using probabilistic models that incorporate prior knowledge and observed data to predict the likelihood of test case failures. This method is particularly effective in environments with limited training data [12].

D. Bayesian Modeling Results The following result is taken directly from the associated study by Chalmers University [12]: The Bayesian modeling approach reported an APFD of 0.72 on the Paint Control dataset.

E. Advanced Neural Network Models In comparison, the neural network models developed in this thesis have shown varying performance across datasets:
Paint Control Dataset:

- **CNN Model** Achieved an APFD of 0.78 and an NAPFD of 0.62.
- **Dueling Network Model** Attained an APFD of 0.80 and an NAPFD of 0.74.
- **Transformer Model** Displayed an APFD of 0.80 with an NAPFD of 0.76.

IOF/ROL Dataset:

- **CNN Model** Achieved an APFD of 0.64 and NAPFD of 0.50.
- **Dueling Network Model** Attained an APFD of 0.65 and NAPFD of 0.51.
- **Transformer Model** Displayed an APFD if 0.64 and NAPFD of 0.49.

F. Summary The varied performance across these models underscores the importance of selecting the appropriate machine learning strategy based on specific test environment needs. The higher APFD and NAPFD scores achieved by the Dueling Network and Transformer models suggest that these architectures are particularly adept at managing the complexities of test case data that involve intricate dependencies or require understanding of long-range patterns in the data sequence.
Moreover, the superior results from the CNN, Dueling Network, and Transformer models compared to the DeepOrder and Bayesian approaches on the Paint Control dataset indicate the potential benefits of employing advanced neural network architectures that can learn and adapt more effectively to the nuances of specific datasets. However, the results from the IOF/ROL dataset highlight the need for further refinement of these models to handle different data characteristics effectively.

6 Discussion

This section provides an in-depth analysis of the empirical findings detailed in Section 5, examining the performance of advanced ML models—CNNs, Transformers, and Dueling Networks—in enhancing TCP within CI environments under various operational conditions. This analysis draws upon underlying machine learning theories and practical considerations to explain observed performances and their implications.

6.1 Influence of Hidden Layer Sizes

The minimal impact of increasing hidden layer sizes on the APFD scores across models, as shown in the IOF/ROL, Paint Control, and Plejd datasets, suggests that the models achieve a threshold of feature extraction capacity beyond which additional complexity does not translate to performance gains. This aligns with the theory of diminishing returns in neural network depth, where beyond a certain depth, additional layers can lead to overfitting or fail to contribute to model learning, particularly in datasets of limited size or complexity [31]. These findings are consistent with the observations made in the RETECS paper, where increasing the number of hidden nodes did not lead to a performance gain.
Increasing the number of hidden nodes in a neural network does not always lead to a performance gain due to several factors:

- **Overfitting:** A network with too many hidden nodes may have excessive capacity relative to the complexity of the task or the size of the training data. This can lead to learning noise and random fluctuations in the training data, resulting in great performance on the training data but poor generalization to new, unseen data [31, 34, 35].
- **Diminishing Returns:** After a certain point, adding more hidden nodes yields minimal improvement in the network's problem-solving capability. The network already has sufficient capacity to capture essential patterns in the data, resulting in diminishing returns [31, 34].
- **Increased Computational Complexity:** More hidden nodes mean more parameters to train, increasing the computational burden. This leads to longer training times and requires more data to effectively train the additional parameters without overfitting [36].
- **Vanishing or Exploding Gradients:** Gradients used in the training process can become very small (vanish) or very large (explode) as they propagate back through the layers during training, making learning unstable or slow, particularly when simple backpropagation with basic activation functions is used [37, 32].

In the context of this study, the findings across the IOF/ROL, Paint Control, and Plejd datasets reinforce the notion that adding more hidden nodes does not significantly improve performance in APFD scores. Specifically, the CNN, Dueling Network, and Transformer models showed consistent APFD scores regardless of hidden layer size, suggesting that these architectures already have adequate capacity to handle the datasets. However, the MLP model exhibited some sensitivity to hidden layer size, with performance fluctuations observed in the Paint Control and Plejd datasets.
Overall, these results highlight the importance of balancing model complexity with the size and complexity of the dataset. The consistent performance observed for most neural network architectures suggests that feature extraction capacity has reached a saturation point, beyond which additional complexity does not contribute to further performance gains.

6.2 Effect of History Length

The investigation into the impact of history length on model performance across various datasets provides nuanced insights into the adaptability of different neural network architectures in TCP tasks. Unlike the MLP network and Tableau agent, which exhibit sensitivity to changes in history length, the CNN, Dueling Network, and Transformer models demonstrate a remarkable robustness, maintaining consistent performance irrespective of the history length variations.
This robustness in CNNs, Dueling Networks, and Transformers can be attributed to their inherent architectural advantages. CNNs are particularly effective at capturing and generalizing key features from sequences of data, regardless of their length [34]. This ability suggests that CNNs can handle varied input sizes without significant performance degradation, making them highly effective in environments where test cases may not consistently follow similar historical patterns.
Dueling Networks and Transformers, on the other hand, are designed to process information dynamically, which contributes to their stability across different history lengths. Transformers, with their attention mechanisms, are adept at focusing on relevant parts of the input data [11], a feature that becomes particularly valuable in managing the temporal dependencies in TCP tasks. Similarly, Dueling Networks' architecture, which separately assesses the state and the value of actions, allows them to maintain performance by effectively disentangling the necessary historical context needed for decision-making from the broader input data [10].
These characteristics highlight a critical aspect of neural network design for TCP in CI environments: the

capability to adapt to varying amounts of historical data is paramount. This adaptability ensures that the models are not only sensitive to the presence of relevant information but are also capable of disregarding redundant or irrelevant data, thus optimizing the prioritization process. The contrast in performance stability between CNNs, Dueling Networks, Transformers, and the more traditional MLP or heuristic-based Tableau agent further underscores the importance of selecting appropriate model architectures based on the specific requirements and variability of the CI environment [31]. These findings affirm the theoretical perspectives on the importance of data representation and model architecture in learning systems, as discussed extensively in machine learning literature. The ability of CNNs, Dueling Networks, and Transformers to maintain efficacy across varied history lengths without the need for extensive re-tuning or modification suggests a significant advancement in the application of neural networks to TCP. This adaptability not only enhances the practical utility of these models in dynamic software development scenarios but also offers a robust framework for further research into optimizing neural network architectures for complex, data-driven tasks in software engineering [38].

6.3 Time Constraints and Scheduling

The data presented in Figures 8, 9, and 10 highlight the effectiveness of different models in scheduling test cases under time constraints. While an improvement in NAPFD scores across all models with increasing scheduling time ratios is expected due to the flexibility to include more test cases, the real insight lies in how efficiently each model prioritizes test cases when constrained by lower time ratios.

The Dueling Network model consistently shows superior performance across most time ratios. Its architecture, which separately estimates state values and action advantages, allows it to efficiently identify and prioritize failing test cases early, maximizing fault detection even with limited scheduling time. However, the Transformer model excels at higher scheduling time ratios, demonstrating its ability to focus on relevant features and adjust prioritization effectively as more time is available. Both models achieve comparable scores at the highest ratios, indicating their robustness in prioritizing critical test cases when time constraints are relaxed.

The CNN model, while generally outperforming traditional heuristic methods, demonstrates consistent improvement as the scheduling time ratio increases. This trend suggests that while convolutional architectures can identify significant patterns in test data, their prioritization effectiveness benefits greatly from additional scheduling time. Despite this, the CNN model still lags behind Dueling Networks and Transformers in scheduling efficiency at lower time ratios.

The results highlight the importance of scheduling flexibility in CI environments. Models with high computational capacity, like Dueling Networks and Transformers, can dynamically adjust prioritization strategies based on available testing time, effectively focusing on the most critical test cases. This flexibility ensures that software defects are detected early, maintaining high-quality standards while accommodating the rapid deployment cycles demanded by agile development practices. Moreover, the trend observed across datasets suggests that integrating strategies allowing for adjustable scheduling time ratios can lead to enhanced fault detection without compromising the agility of software development processes. Such strategies enable CI systems to balance rapid deployment with thorough testing, dynamically adjusting the amount of testing based on the criticality of each release and the available time window.

In conclusion, our analysis supports the idea that neural network models like Dueling Networks and Transformers significantly enhance TCP, particularly under stringent time constraints. This aligns with modern software engineering practices emphasizing flexibility and efficiency in testing strategies to ensure high software quality in agile environments. While higher scheduling time ratios undoubtedly improve NAPFD scores, the ability of these advanced models to prioritize test cases effectively under lower ratios reinforces their importance in continuous integration pipelines.

6.4 Effects of Reward Functions

The effectiveness of different reward functions in TCP significantly influences the performance of machine learning models within CI environments. By employing the APFD metric to analyze the behavior of CNN, Dueling Network, and Transformer models across three datasets, this section examines the impact of reward functions on their performance.

The CNN model exhibits stable APFD scores across different reward functions and datasets. Notably, in the Plejd and ABB IOF/ROL datasets, the CNN demonstrates consistent prioritization performance (Figure 11). This consistency aligns with the known strengths of CNN architectures in extracting spatial patterns, enabling the model to generalize well across varied conditions. Although the CNN does not always outperform other models, it shows robust adaptability to different reward functions, suggesting that its generalization capability makes it suitable for various CI environments.

The Dueling Network model outperforms other models under specific reward functions, particularly in the ABB Paint Control dataset with the Tcfail function (Figure 12). This model is adept at prioritizing failing test cases early, leading to improved APFD scores compared to the other models. Its dual-path architecture allows it

to separately estimate state values and the advantages of actions, thus effectively adapting to reward functions that emphasize fault detection. The superior performance under the Tcfail function suggests that Dueling Networks are well-suited for environments prioritizing immediate fault detection.

The Transformer model displays consistent performance across all datasets and reward functions, particularly excelling under the Timerank Reward function (Figure 13). The attention mechanisms inherent to Transformers enable them to identify temporal dependencies and focus on the most relevant test cases. This model maintains competitive APFD scores, closely aligning with the Dueling Network under Timerank, highlighting its ability to handle varying temporal patterns in test data effectively. The stability across reward functions indicates that Transformers can generalize well in different CI scenarios. The analysis of these models under varying reward functions reveals key insights into their comparative strengths. The CNN model provides balanced performance across different rewards, making it a reliable choice in unpredictable environments. The Dueling Network excels with Tcfail, highlighting its advantage in fault prioritization scenarios requiring early detection. Meanwhile, the Transformer model maintains robust performance, making it ideal for CI environments needing stability and reliability.

The studies by Mirzaei et al. [39] and Han et al. [40] reinforce the significance of reward functions in R)-based TCP, indicating that reward functions using historical and current information impact prioritization strategies differently. Historical information-based rewards provide a more comprehensive assessment of failure detection capacity, leading to better performance in fault detection. Conversely, current information-based rewards, like Tcfail, emphasize recent failures and can be advantageous in dynamic environments where recent test results are more indicative of future faults. Mirzaei et al.'s analysis suggests that time-window-based reward functions strike a balance between historical and current data, offering a comprehensive yet dynamic assessment of test case importance. Han et al. further emphasize that RL-based approaches with well-crafted rewards, such as Dueling Networks with Tcfail and Timerank functions, enhance prioritization strategies by directly aligning the agent's objectives with the desired prioritization outcomes.

The variability in model performance across reward functions underscores the importance of selecting appropriate rewards based on specific CI environment requirements. While CNNs offer consistent performance across various rewards, Dueling Networks and Transformers stand out under specific conditions due to their architecture-specific advantages. The findings from Mirzaei et al. and Han et al. reinforce the need for nuanced reward function selection, tailored to the prioritization strategy and goals, to maximize the fault detection efficiency of machine learning models in CI environments.

6.5 Model Comparisons with Heuristic Approaches

The comparative analysis of the performance of the CNN, Dueling Network, and Transformer models provides significant insights into their effectiveness across different CI cycles, particularly when benchmarked against heuristic methods such as Sorting and Weighting. In these analyses, negative APFD scores represent superior model performance over heuristics, whereas positive APFD scores indicate heuristics outperforming the models.

In the Plejd dataset, the Transformer model consistently demonstrated superior performance across different CI cycles, outperforming both the CNN and Dueling Network models. When comparing against the Sorting heuristic, the Transformer model achieved a mean APFD score of -0.0032 at step 90 and -0.0177 at step 120, highlighting a significant advantage over the heuristics. By contrast, the CNN model showed relatively consistent performance, reaching a mean APFD score of -0.0041 at step 90 and 0.0087 at step 120 for the Sorting heuristic. Meanwhile, the Dueling Network model exhibited broader variability, with scores ranging from -0.0264 at step 90 to 0.0072 at step 120 against the Sorting heuristic. Despite these fluctuations, all three models maintained a consistent advantage over the heuristics in later steps, particularly in step 270, where the Transformer model achieved a peak APFD score of 0.0632, followed by the CNN model at 0.0668.

In the ABB Paint Control dataset, the CNN, Dueling Network, and Transformer models each exhibited varying degrees of superiority over the heuristics. Against the Sorting heuristic, the CNN model demonstrated a mean APFD score of -0.0024 at step 120, while the Dueling Network model showed slightly better performance with a mean APFD score of -0.0039. The Transformer model, however, achieved a comparable APFD score of -0.0066 at the same step. In later CI cycles (steps 240 to 300), the CNN model established a strong lead, with a mean APFD score of 0.0346 at step 240 and 0.0464 at step 270, outperforming both the Dueling Network and Transformer models. The Dueling Network model trailed closely behind, achieving mean APFD scores of 0.0237 at step 240 and 0.0376 at step 270 against the Sorting heuristic. Despite these variations, the Transformer model remained highly competitive, achieving mean APFD scores of 0.0374 and 0.0632 at steps 240 and 270, respectively.

The ABB IOF/ROL dataset presented a unique challenge, with all three models showing significant variability against the heuristics. Against the Sorting heuristic,

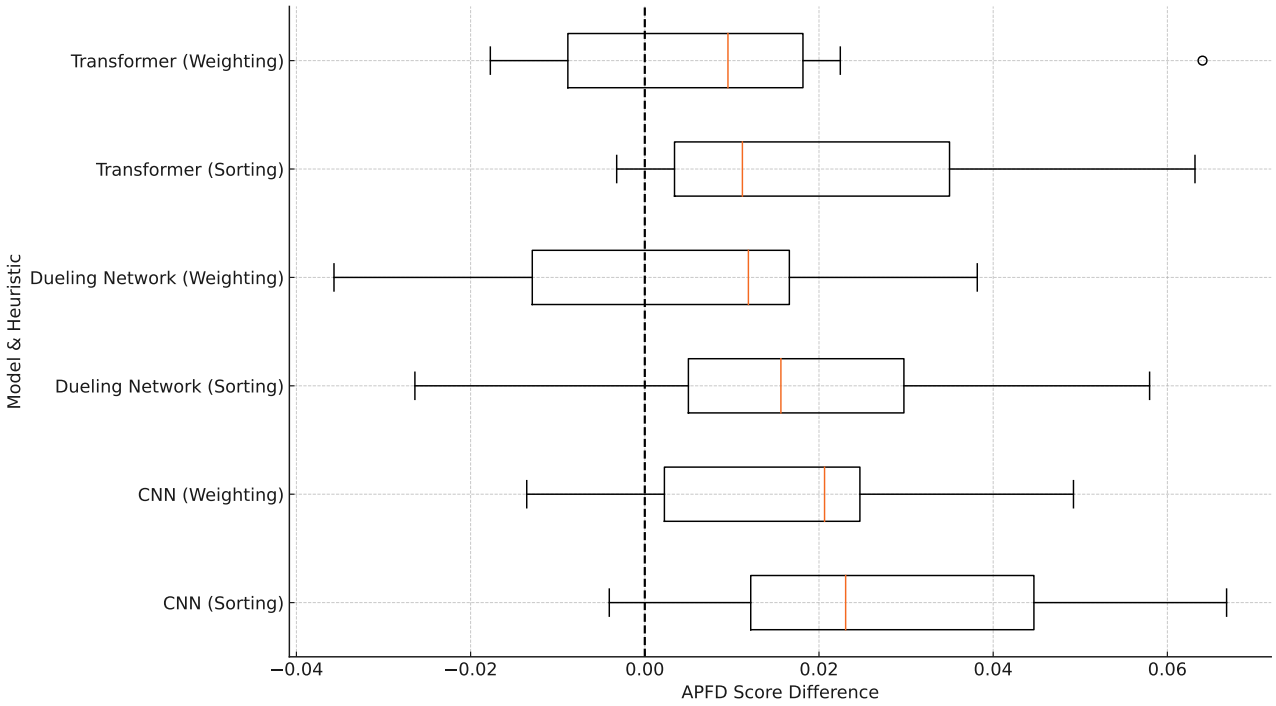

Figure 18: Boxplot of the APFD Score Difference and Variance for CNN, Dueling Network, and Transformer Models by Heuristic.

the CNN model maintained relatively consistent performance, with mean APFD scores ranging from -0.0134 at step 90 to 0.0231 at step 210. The Dueling Network model, however, struggled in comparison, achieving scores as low as -0.0264 at step 90 but recovering to 0.0072 at step 120 and reaching 0.0349 at step 240. The Transformer model displayed broader variability, with APFD scores ranging from -0.0125 at step 90 to 0.0632 at step 270, outperforming the CNN and Dueling Network models in later CI cycles.

The analysis of variance in APFD scores across models and heuristics revealed distinct trends in performance, as shown in Figure 18. The box plots indicated that the CNN model exhibited the narrowest interquartile range, particularly for the Sorting heuristic, suggesting consistent performance. The Dueling Network model demonstrated broader variability, indicating mixed performance relative to heuristics. The Transformer model, while exhibiting the widest range of scores, maintained a median score below zero, particularly for the Weighting heuristic, signifying superior performance over the heuristic.

Figure 19 presents the mean APFD scores across steps for the CNN, Dueling Network, and Transformer models, grouped by the heuristic approaches (Sorting and Weighting). The results highlight that the Transformer model generally outperforms other models in later CI cycles, demonstrating its adaptability and efficiency. The CNN model maintained consistent performance with relatively narrow variance in scores, while the Dueling Network model exhibited broader variability, indicating mixed performance relative to heuristics.

In conclusion, the comparative analysis of the CNN, Dueling Network, and Transformer models across different datasets and heuristics reveals distinct performance characteristics. The Transformer model generally outperformed the other models in later CI cycles, demonstrating its adaptability and efficiency. The CNN model maintained consistent performance with relatively narrow variance in scores, while the Dueling Network model exhibited broader variability, indicating mixed performance relative to heuristics. These observations underscore the importance of model selection and tuning in achieving optimal performance in Continuous Integration cycles.

6.6 Comparison of Advanced Neural Network Models with DeepOrder and Bayesian Models

The comparison of advanced neural network models with existing prioritization techniques, specifically DeepOrder and Bayesian modeling, reveals several interesting insights into their effectiveness and applicability in software testing. The discussion focuses on interpreting the results achieved by CNN, Dueling Network, and Transformer models relative to DeepOrder and Bayesian modeling approaches.

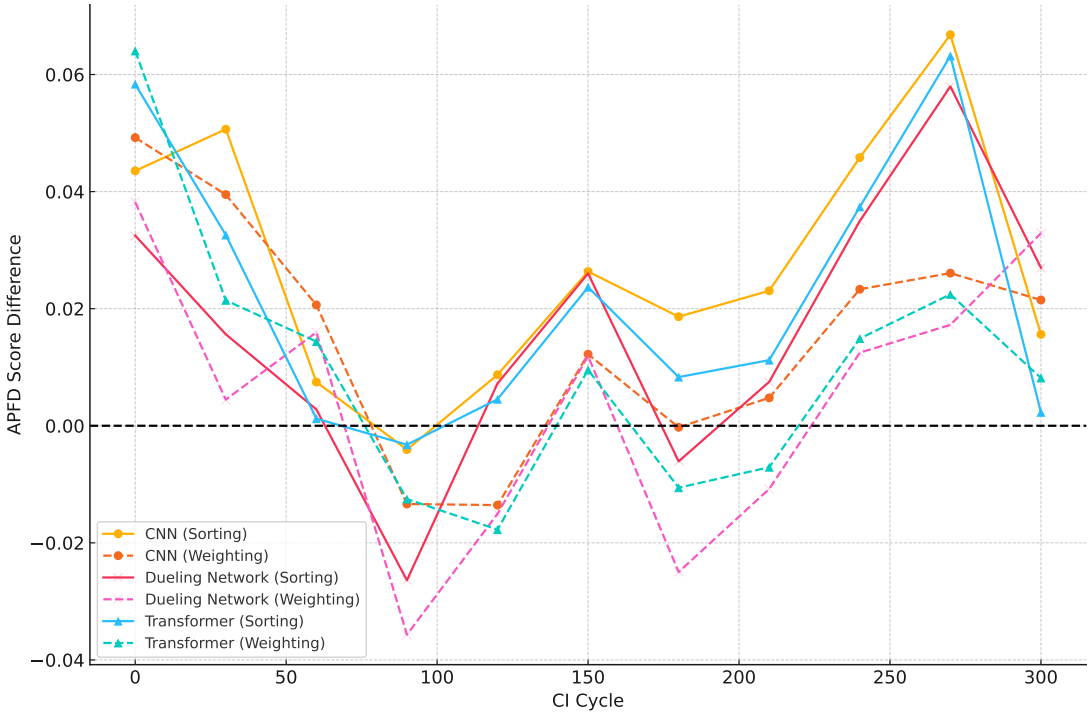

Figure 19: Mean APFD Scores for CNN, Dueling Network, and Transformer Models by Heuristic.

The DeepOrder model, as documented by Sharif et al. [13], achieved APFD scores of 0.76 and 0.67 on the Paint Control and IOF/ROL datasets, respectively. However, when time constraints are considered, the NAPFD scores drop to 0.52 and 0.56, highlighting a reduction in prioritization performance. This reduction suggests that while DeepOrder is capable of prioritizing test cases effectively under normal conditions, its performance diminishes when the available time is restricted. The Bayesian modeling approach, which is more tailored to small to medium-sized enterprises, reported an APFD of 0.72 on the Paint Control dataset [12]. Although the APFD score is slightly lower than DeepOrder, the Bayesian method remains effective in prioritizing test cases in environments with limited or highly specific data. However, its generalizability to other datasets remains uncertain due to a lack of reported performance metrics on other datasets.

The neural network models developed in this thesis have demonstrated varying levels of performance across the Paint Control and IOF/ROL datasets. On the Paint Control dataset, the CNN model achieved an APFD of 0.78 and an NAPFD of 0.62, indicating its superior ability to prioritize test cases even when time constraints are in place. This suggests that CNNs are better suited for environments where rapid prioritization is critical, providing a higher APFD score than both DeepOrder (0.76) and Bayesian modeling (0.72). The Dueling Network model further improved these metrics, reaching an APFD of 0.80 and an NAPFD of 0.74. Its robustness in handling complex prioritization scenarios is evident, as it surpasses DeepOrder and Bayesian modeling approaches in both metrics. Achieving an APFD of 0.80 and an NAPFD of 0.76, the Transformer model showcases its ability to adapt to the nuances of test data. Its focus on relevant features enables it to prioritize test cases effectively, outperforming DeepOrder and Bayesian modeling.

On the IOF/ROL dataset, the CNN model achieved an NAPFD of 0.50, demonstrating its capacity to prioritize test cases. However, it fell short compared to DeepOrder, which attained an NAPFD of 0.56. The Dueling Network model improved upon the CNN, achieving an NAPFD of 0.51 but still not outperforming DeepOrder. The Transformer model displayed an NAPFD of 0.49, indicating challenges in prioritizing test cases effectively on this dataset, potentially due to the specific nature of the test case data.

The comparative analysis of the neural network models against DeepOrder and Bayesian modeling methods pro-

vides several insights. The Dueling Network and Transformer models significantly outperformed DeepOrder and Bayesian modeling approaches on the Paint Control dataset, achieving higher APFD and NAPFD scores. This superior performance underscores their capacity to handle intricate prioritization tasks in dynamic CI environments effectively. The neural network models demonstrated mixed results on the IOF/ROL dataset. Although they showcased better prioritization than traditional methods in certain aspects, none managed to outperform DeepOrder comprehensively. This result highlights the need for further refinement of these models to handle specific dataset characteristics effectively. The higher APFD and NAPFD scores achieved by the Dueling Network and Transformer models suggest that these architectures are adept at managing the complexities of test case data involving intricate dependencies or requiring an understanding of long-range patterns. The results imply that advanced neural network models, such as Dueling Networks and Transformers, offer substantial benefits in CI environments where test cases evolve rapidly. Their ability to quickly adjust prioritization strategies can significantly enhance the efficiency of the software development lifecycle.

Overall, this analysis emphasizes the importance of selecting appropriate machine learning strategies for TCP based on the specific requirements of the CI environment.

6.7 Theoretical and Practical Implications

The findings from this study significantly contribute to the theoretical understanding of neural network architectures in TCP and reinforce their practical application within CI environments.

Theoretically, this research enhances the understanding of how neural network architectures can be effectively optimized for dynamic and data-intensive applications such as TCP. The comparative analysis of the CNN, Dueling Network, and Transformer models against heuristic methods and contemporary models like DeepOrder and Bayesian approaches demonstrates that advanced machine learning models can improve the efficacy of test case scheduling and prioritization [41]. Specifically, the CNN model's APFD of 0.78 and the Transformer model's APFD of 0.80 on the Paint Control dataset illustrate that neural networks can be optimized to detect faults early in the testing cycle, even under time constraints. Moreover, the Dueling Network model's robust APFD score of 0.80 and NAPFD score of 0.74 demonstrate its capacity to handle complex prioritization scenarios. These models outperform DeepOrder, which achieved an APFD of 0.76, and the Bayesian modeling approach, which had an APFD of 0.72 on the Paint Control dataset, indicating that neural networks are adept at managing intricate test case data involving intricate dependencies and long-range patterns.

Additionally, this study extends the discourse beyond traditional heuristic-based approaches, emphasizing that machine learning models, with their ability to handle complex and variable inputs, offer a transformative potential in automating and optimizing TCP processes within CI frameworks. The Dueling Network and Transformer models, particularly with their higher APFD scores, demonstrated resilience and adaptability under varying reward conditions, suggesting that advanced neural networks can adaptively learn and prioritize based on the intricacies of the data provided. This resilience is evident in the Dueling Network's APFD lead of 0.03 over DeepOrder on the Paint Control dataset and its consistent performance across different reward functions. Practically, these findings advocate for a nuanced integration of machine learning models into CI pipelines. The stable performance of the Transformer model across diverse reward conditions, achieving an NAPFD of 0.76 on the Paint Control dataset, makes it particularly suitable for environments where time restrictions are of priority. The superior performance of the Dueling Network model further underscores the practical implications of leveraging advanced architectures in dynamic CI environments. The CNN, Dueling Network, and Transformer models consistently outperformed traditional heuristic methods like Sorting and Weighting, especially in later CI cycles in the Plejd dataset. This consistency suggests that the operational success of machine learning models in CI pipelines is contingent upon careful configuration and tuning.

Moreover, this study encourages organizations to consider not only the immediate benefits of integrating ML models into their CI pipelines but also the long-term adaptability of these systems. For instance, the consistent performance of the Transformer model across datasets suggests that organizations can maintain efficiency in detecting faults even as software evolves and testing conditions change. As machine learning in software testing evolves, it becomes critical to continually assess and recalibrate these models to maintain their effectiveness under shifting testing conditions and software evolution. This comprehensive approach not only augments the theoretical foundations of using neural networks in TCP but also provides a clear roadmap for their practical deployment, ensuring that the integration of these technologies enhances rather than hinders the agile and dynamic nature of modern software development processes.

The results indicate that the integration of these neural network models into existing CI pipelines can significantly improve the efficiency and effectiveness of the testing process, ensuring that feedback is provided to developers swiftly and accurately. While models like DeepOrder and Bayesian modeling methods remain ef-

fective in specific contexts, the adaptability and robustness of proposed architectures underscore the transformative potential of machine learning in software testing. Organizations that invest in these advanced models can expect enhanced testing efficiency, particularly in rapidly evolving CI environments where test cases must be continuously prioritized and updated.

6.8 Future Directions and Concluding Remarks

Future research should delve into the development and integration of hybrid models that synthesize the strengths of CNNs, Transformers, and Dueling Networks. Such models have the potential to leverage their respective advantages—CNNs for their feature extraction capabilities, Transformers for their attention mechanisms in handling sequential data, and Dueling Networks for their effective differentiation between state values and advantages in decision-making processes. This integration could yield more robust solutions for dynamic and complex TCP tasks.

Moreover, the application of transfer learning could significantly enhance the adaptability and efficiency of these models. Pre-trained models could be fine-tuned for specific TCP tasks, leveraging learned representations from similar domains to accelerate convergence and improve fault detection accuracy [42]. This approach could reduce training time and resources while improving performance, especially in environments with limited labeled data. Additionally, future research should explore using Transformers to handle dependencies between test cases within an entire test suite. Prioritization techniques can be classified based on their granularity: fine-grained and coarse-grained. Fine-grained prioritization typically focuses on individual test cases, which is used in this study, whereas coarse-grained prioritization addresses groups of test cases or modules. According to Elbaum et al. (2002) [43], granularity significantly affects the performance of prioritization techniques. Instead of focusing solely on features within individual test cases (coarse-grained), this broader perspective could uncover intricate relationships across test cases, allowing for more informed prioritization decisions. By modeling the entire test suite as a sequence, Transformers can identify long-range dependencies and predict the impact of executing one test case on others, ultimately enhancing prioritization strategies. Moreover, the APFD metric could be used as a reward function in itself in future studies if a coarse-grained approach is adopted. In this study, using APFD as a reward function was attempted but yielded suboptimal results due to the fine-grained approach, where models handle one test case at a time. Adopting a coarse-grained approach in future research could allow the models to optimize prioritization over the entire test suite, potentially improving fault detection and resource allocation in a more holistic manner.

Experimental studies across a broader spectrum of CI environments are essential. These studies would help validate the generalizability of the findings presented and refine model configurations to cater to the specific needs of diverse application scenarios. By testing these models in various real-world settings, researchers can also identify new challenges and opportunities for further enhancements in machine learning-based TCP techniques.

6.9 Threats to Validity

In evaluating the robustness of these findings, it is essential to consider several potential threats to validity that could impact the conclusions of this study.

Internal Validity: The first threat to internal validity is the influence of random decisions on the results. To mitigate this threat, the experiments were repeated 20 times, and the averaged results are reported. Another potential threat is related to the existence of faults within the implementation. This was approached by using well-established frameworks like PyTorch to develop the models and ensure the reliability of the implementation. Finally, many machine learning algorithms are sensitive to their parameters, and a feasible parameter set for one problem environment might not work as well for a different one. During the experiments, the initially selected parameters were not changed across different problem environments to allow better comparison. These parameters were chosen based on a combination of literature review and preliminary testing. In real-world settings, these parameters can be adjusted to fine-tune the approach for specific environments.

External Validity: The results presented in this study are promising, but their generalizability across different CI environments, software projects, and industries has not been fully tested. Our evaluation is based on three industrial datasets, including those provided by the RETECS paper, which limits the results' applicability given the wide variety of CI environments and failure distributions. According to the RETECS paper, one of the datasets had only been used in a single publication and different settings. To enhance external validity, future studies should replicate these experiments across diverse domains and integrate findings from various software development contexts to confirm the models' utility in broader operational settings. Further research should also focus on acquiring more publicly available datasets to enrich the analysis and facilitate more comprehensive evaluations of machine learning models in TCP.

Construct Validity: It is crucial that the metrics used to evaluate model performance accurately reflect the objectives of TCP. A misalignment between the

chosen evaluation metrics and the practical goals of TCP could lead to misleading conclusions. This study ensured that metrics were carefully selected to align with the real-world demands of software testing, enhancing the study's relevance and applicability. The APFD metric was selected due to its widespread adoption in TCP research and its comparability with other studies. However, the APFD metric has inherent flaws that could affect the interpretation of results. First, the APFD metric assumes that all faults have the same severity, giving equal weight to trivial and critical faults. This can lead to prioritizing test cases that detect less critical faults over those that identify highly severe ones. In real-world scenarios, critical faults can have a significant impact on system reliability and user satisfaction, whereas trivial faults might not warrant immediate attention. Thus, a metric that does not differentiate fault severity might fail to align test prioritization with actual risk levels, leading to suboptimal decision-making and resource allocation in software testing. Nayak et al. (2016) [44] and Elbaum et al. (2001) [45] highlighted this limitation and proposed alternatives that incorporate fault severity into prioritization. Second, APFD does not consider the varying costs associated with executing different test cases. Test cases that require significantly more time and resources are treated the same as low-cost tests, which can skew prioritization results. For instance, a time-consuming test case might detect multiple critical faults, but if a shorter test case can detect the same faults with less resource expenditure, prioritizing the longer test case could be inefficient. Ignoring the costs associated with test execution can lead to higher overall testing costs and longer testing cycles, which are critical factors in real-world software development where time and resources are limited. Elbaum et al. (2001) introduced the APFDc metric to address this issue by accounting for varying test case costs. Third, APFD does not distinguish between faults that have already been addressed versus newly introduced ones. This limitation could result in skewed prioritization results, especially in regression testing contexts where identifying new faults is crucial. In the context of regression testing, the primary goal often shifts towards detecting newly introduced faults that might have been inadvertently created while fixing old ones or adding new features. A metric that does not differentiate between old and new faults might give undue credit to test cases that repeatedly identify previously known faults, thus failing to prioritize tests that would uncover new issues introduced in the latest code changes. Fourth, the metric assumes that the order in which test cases are executed does not affect the detection of faults. However, in practice, the sequence in which test cases are run can influence the effectiveness of fault detection. For example, running a critical test early might identify faults that, if not fixed, could render subsequent tests ineffective or irrelevant. Conversely, certain tests might be more effective when executed after others due to dependencies or interactions within the system under test. Thus, ignoring the execution order can overlook strategic benefits in test scheduling and potentially reduce the overall effectiveness of the testing process. Nayak et al. (2016) demonstrated how different prioritization sequences could significantly impact testing outcomes. Despite these limitations, the APFD metric remains the standard benchmark for comparing TCP effectiveness across studies due to its ease of calculation and comparability with prior research. However, acknowledging its inherent flaws and using complementary metrics are essential for validating results. To mitigate the flaws of the APFD metric, additional metrics like Defect Detection Rate and Test Cost/Time to Test can provide a more comprehensive evaluation of prioritization effectiveness. Furthermore, alternative metrics like APFDc (Elbaum et al., 2001) and Weighted Priority (Nayak et al., 2016) that account for fault severity and test execution costs offer more nuanced insights into TCP effectiveness.

Conclusion Validity: The reliability of the conclusions is contingent upon the robustness of the statistical methods and analyses employed. In this research, statistical analyses like mean performance comparisons and standard deviation were utilized to ensure the integrity of data analysis. The experiments were repeated 20 times, and the averaged results were reported to minimize the impact of random variations. To address potential threats and verify the robustness of the findings, future research should expand the scope of the experiments to include a broader range of environments and refine the models and methods based on comprehensive empirical data. Such endeavors will better ascertain the transformative potential of machine learning in TCP, ultimately contributing to more agile, efficient, and effective software testing processes in CI environments.

7 Conclusion

This thesis embarked on a comprehensive exploration of integrating advanced neural network architectures. The study was driven by the imperative to move beyond traditional heuristic-based approaches, employing machine learning techniques to enhance the dynamics, efficiency, and effectiveness of regression testing in modern software development.

7.1 Synthesis of Empirical Findings

The research confirmed that each neural network model brings significant improvements to the TCP process, leveraging unique capabilities:

- **CNNs** were pivotal in recognizing complex spatial and temporal patterns in test data, thereby facilitating the identification of test cases that are more likely to expose faults. Their effectiveness in streamlining the testing process aligns with the pressing needs of modern CI environments, which demand rapid yet accurate testing cycles.
- **Transformers** excelled in their ability to process sequences of test cases, utilizing advanced attention mechanisms. This model was instrumental in adapting TCP to the nuanced requirements of evolving codebases, particularly in settings where changes occur rapidly and frequently, necessitating a dynamic approach to test prioritization.
- **Dueling Networks** offered a significant breakthrough with their dual-path architecture, which enhances decision-making by evaluating the potential impact of each test case independently. This model's performance underscored its utility in optimizing test sequences to quickly uncover the most critical faults, thus supporting faster resolution times and maintaining high software quality.

7.2 Limitations and Comprehensive Future Research Directions

While the findings of this thesis demonstrate significant advancements in the application of neural networks to TCP within CI environments, they are accompanied by inherent limitations that present opportunities for future research:

- **Scalability and Performance Optimization:** The current models show promise in controlled settings, yet their scalability to larger, more complex CI environments remains to be fully explored. Future research should focus on enhancing the scalability and performance optimization of these models, ensuring their efficiency and practicality for widespread adoption in diverse and demanding real-world scenarios.
- **Integration Challenges:** Integrating advanced machine learning models into established CI pipelines poses significant technical and cultural challenges. Further studies are needed to develop and refine integration strategies that minimize disruption and maximize the utility of these models within existing software development workflows.

Addressing these limitations and exploring these future research directions are critical for realizing the full potential of machine learning in enhancing TCP practices. Continued innovation and research in this area will help in developing more agile, efficient, and effective software testing processes, thereby supporting the rapid evolution of software development practices in the CI landscape.

7.3 Concluding Remarks

The theoretical contributions of this thesis extend the boundaries of machine learning application in software testing, offering a new perspective on the potential of advanced neural networks in TCP within CI environments. This research has demonstrated not just the feasibility but the profound advantages of employing sophisticated models like CNNs, Transformers, and Dueling Networks to enhance the dynamics, efficiency, and effectiveness of regression testing. Practically, the integration of these models into TCP practices represents a paradigm shift in traditional testing methodologies. It promises to revolutionize software testing by delivering unprecedented speed, efficiency, and precision. These qualities are crucial in modern CI environments, where the demand for rapid yet accurate testing cycles continues to grow. The practical implications of this study underscore the potential for these models to transform TCP from a predominantly heuristic-driven practice to one that is more data-driven and analytics-based.
In conclusion, this thesis not only highlights the viability of using sophisticated neural network architectures in enhancing TCP but also sets the stage for further innovation in software testing practices. As the field of software development moves towards more agile and integrated practices, the role of intelligent automation in testing will expand, reinforcing the need for ongoing research and development in this area. Future studies should explore the integration of hybrid models and extend the application of these findings across various software environments to fully realize the potential of AI-driven solutions in meeting the complex demands of modern software development.

Bibliography

[1] P. M. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*. Boston, MA: Pearson Education, 2007, first printing, June 2007.

[2] H. Leung and L. White, "Insights into regression testing (software testing)," in *Proceedings. Conference on Software Maintenance - 1989*, 1989, pp. 60–69. [Online]. Available: https://doi.org/10.1109/ICSM.1989.65194

[3] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001. [Online]. Available: https://doi.org/10.1109/32.962562

[4] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability*, vol. 22, 03 2012. [Online]. Available: https://doi.org/10.1002/stvr.430

[5] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, "Adaptive random testing: The art of test case diversity," *The Journal of Systems and Software*, vol. 83, pp. 60–66, 2010. [Online]. Available: https://doi.org/10.1016/j.jss.2009.02.022

[6] P. Konsaard and L. Ramingwong, "Total coverage based regression test case prioritization using genetic algorithm," in *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2015, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ECTICon.2015.7207103

[7] A. Bajaj and O. P. Sangwan, "A systematic literature review of test case prioritization using genetic algorithms," *IEEE Access*, vol. 7, pp. 126 355–126 375, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2938260

[8] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2017, p. 12. [Online]. Available: https://doi.org/10.1145/3092703.3092709

[9] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012. [Online]. Available: https://doi.org/10.1145/3065386

[10] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2016.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[12] A. Eliasson, "Bayesian modeling approach to test case prioritization," https://odr.chalmers.se/items/f44c16f0-1cb2-45ae-84ae-590d4f818193, 2023.

[13] A. Sharif, D. Marijan, and M. Liaaen, "Deeporder: Deep learning for test case prioritization in continuous integration testing," 2021.

[14] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, no. 3, p. 270–285, jul 1993. [Online]. Available: https://doi.org/10.1145/152388.152391

[15] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Software Testing, Verification and Reliability*, vol. 12, no. 4, pp. 219–249, 2002. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.256

[16] G. Rothermel and M. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 529–551, 1996. [Online]. Available: https://doi.org/10.1109/32.536955

[17] X. Qu, M. B. Cohen, and K. M. Woolf, "Combinatorial interaction regression testing: A study of test case generation and prioritization,"

in *2007 IEEE International Conference on Software Maintenance*, 2007, pp. 255–264. [Online]. Available: https://doi.org/10.1109/ICSM.2007.4362638

[18] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[19] A. Bertolino, A. Guerriero, B. Miranda, R. Pietrantuono, and S. Russo, "Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 1–12.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book.html

[21] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[22] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[23] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, 1992. [Online]. Available: https://doi.org/10.1007/BF00992699

[24] Z. Zhou, P. Zhu, Z. Zeng, J. Xiao, H. Lu, and Z. Zhou, "Robot navigation in a crowd by integrating deep reinforcement learning and online planning," 2021.

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: https://doi.org/10.1038/nature16961

[26] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. [Online]. Available: https://doi.org/10.1109/MSP.2017.2743240

[27] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," 2017.

[28] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017. [Online]. Available: https://arxiv.org/abs/1701.07274

[29] H. Ahmed and A. K. Nandi, *Artificial Neural Networks (ANNs)*. John Wiley & Sons Ltd, 2019, pp. 239–258. [Online]. Available: https://doi.org/10.1002/9781119544678.ch12

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://doi.org/10.1109/5.726791

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[35] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," 2017.

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[37] G. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, vol. 9, no. 1, p. 926, 2012.

[38] Y. Bengio, *Learning Deep Architectures for AI*. Now Foundations and Trends, 2009, vol. 2, no. 1. [Online]. Available: https://doi.org/10.1561/2200000006

[39] H. Mirzaei and M. R. Keyvanpour, "Reinforcement learning reward function for test case prioritization in continuous integration," in *2022 9th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, 2022, pp. 1–6. [Online]. Available: https://doi.org/10.1109/CFIS54774.2022.9756464

[40] Y. Han, G. Chen, and B. Han, "An improved method for test case prioritization in continuous integration based on reinforcement learning," in *Proceedings of the 3rd International Conference on Management Science and Software Engineering (ICMSSE 2023)*. Atlantis Press, 2023, pp. 958–972. [Online]. Available: https://doi.org/10.2991/978-94-6463-262-0_99

[41] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2019.2962027

[42] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. [Online]. Available: https://doi.org/10.1109/TKDE.2009.191

[43] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[44] S. Nayak, C. Kumar, and S. Tripathi, "Effectiveness of prioritization of test cases based on faults," in *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, 2016, pp. 657–662. [Online]. Available: https://doi.org/10.1109/RAIT.2016.7507977

[45] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, 2001, pp. 329–338. [Online]. Available: https://doi.org/10.1109/ICSE.2001.919106

A Default Parameters for the Reinforcement Learning Agents

Table 4: Parameter Overview

RL Agent	Parameter	Value
All	CI cycle's time limit M	100% $\times T_i$.duration
	History Length	15
Tableau	Number of Actions	100
	Exploration Rate (ϵ)	0.2
Network	Hidden Nodes	32
	Replay Memory	10000
	Replay Batch Size	1000
MLP	Hidden Nodes	32
	Replay Memory	10000
	Replay Batch Size	1000
CNN	Hidden Nodes	32
	Replay Memory	10000
	Replay Batch Size	1000
	Learning Rate	0.0009
	Dropout Probability	30%
Dueling Network	Hidden Nodes	32
	Replay Memory	10000
	Replay Batch Size	1000
	Learning Rate	0.0009
Transformer	Hidden Nodes	32
	Replay Memory	10000
	Replay Batch Size	1000
	Learning Rate	0.0001
	Dropout Probability	10%