THE UNIVERSITY OF
NEWCASTLE
AUSTRALIA

TESTING PLAN

REINFORCEMENT LEARNING FOR Soccer-Playing Robots

Document Control

Version:	2.0
Date:	06/09/2025

Document change control:

Version #	Change Description	Date	Author
1.0	Completed First Draft of Test Plan	28/08/2025	Ali Riyaz
2.0	Completed Finalised Version of Test Plan	06/09/2025	Ali Riyaz

Approved by:

This is an agreement between the Project Manager (i.e., the student) and the Project Client (i.e., a representative of the Business) on the requirements as understood at this specific point in time. Additional signature lines may be added, as appropriate.

Signature: Ali.R Date: 06/09/2025

Name: Ali Riyaz Position: Project Manager

Signature: Khaled S Date: 06/09/2025

Name: Khaled Saleh Position: Project Client

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Project Overview

The project involves the development of a Reinforcement Learning (RL) strategy to replace the rule-based Planning Layer of the current NUbots system, incorporating adaptive decision-making abilities to improve the performance of the soccer-playing robots in dynamic game situations. The system is focused on enhancing dribbling skills, specifically to maintain ball possession and score more goals, which the pre-programmed behaviours struggle with. The system integrates a Python-based RL training pipeline with the existing NUbots C++ architecture through ONNX model conversion and NUClear message-passing framework.

The core deliverables include the following:

- **Custom 2D simulation environment (SoccerEnv):** 2-D simulation environment configured for soccer matches using NUbots robot models and a field (recreating RoboCup conditions) to be used for training. This environment provides sufficient physics realism and environmental factors. It includes configurable opponent behaviours and randomising scenarios for improved transferability and exposes the standard Gymnasium interfaces for RL training integration.
- **RL-Training Pipeline:** Python training pipeline for PyTorch models to be trained in two known RL algorithms implemented in Stable-Baselines3 (SB3): Proximal Policy Optimisation (PPO) and Deep Deterministic Policy Gradient (DDPG). It manages curriculum training, reward calculation, experience collection, model optimisation and includes hyperparameter tuning and experiment tracking functionality.
- **Trained Model Policy:** A PyTorch model policy with neural network weights and configuration files that achieve the target performance metrics in the custom '*SoccerEnv*' environment, which will be used to train models on PPO and DDPG. This model can be transferred to other robot hardware platforms without needing to implement components of the Behaviour system again.
- **ONNX Conversion Pipeline:** Pipeline to convert the trained PyTorch models to ONNX format for C++ runtime compatibility. Due to the random nature of the PPO algorithm, a wrapper (*DeterministicPPOWrapper*) has been developed to exclude the random functions of PPO to produce the same output given the same input. This component includes functionality to validate the accuracy of the outputs from the PyTorch and ONNX models given the same inputs.
- **Quantitative Evaluation Framework:** An evaluation framework to evaluate model performance through performance metrics such as success rate, goals scored, collision and fall frequency, etc. and graph visualisations (i.e. rewards across timesteps, etc.)
- **NUClear Inference Module**: A C++ NUClear module to load the trained policy model with ONNX runtime, allowing for inference communication for decision-making at ≥ 50Hz frequency on NUbots hardware. It integrates with the NUbots system to receive sensor inputs and send strategic commands to the hardware layers.

To establish the scope and complexity of testing required, it is essential to understand how the RL module integrates within the existing NUbots system architecture. The RL module represents a significant architectural change, replacing the rule-based Planning Layer with a machine learning component that must interface with multiple existing subsystems. Figure 1 illustrates the system boundaries and interfaces that require comprehensive testing and validation. It uses C4 modelling at Layer 2 to show how the new RL Module (highlighted in orange) fits within the Behaviour subsystem and identifies the critical integration points that must be validated through testing: sensor data input from Vision and Localisation systems, strategic communication with the Skills layer, and coordination with the overall system workflow.

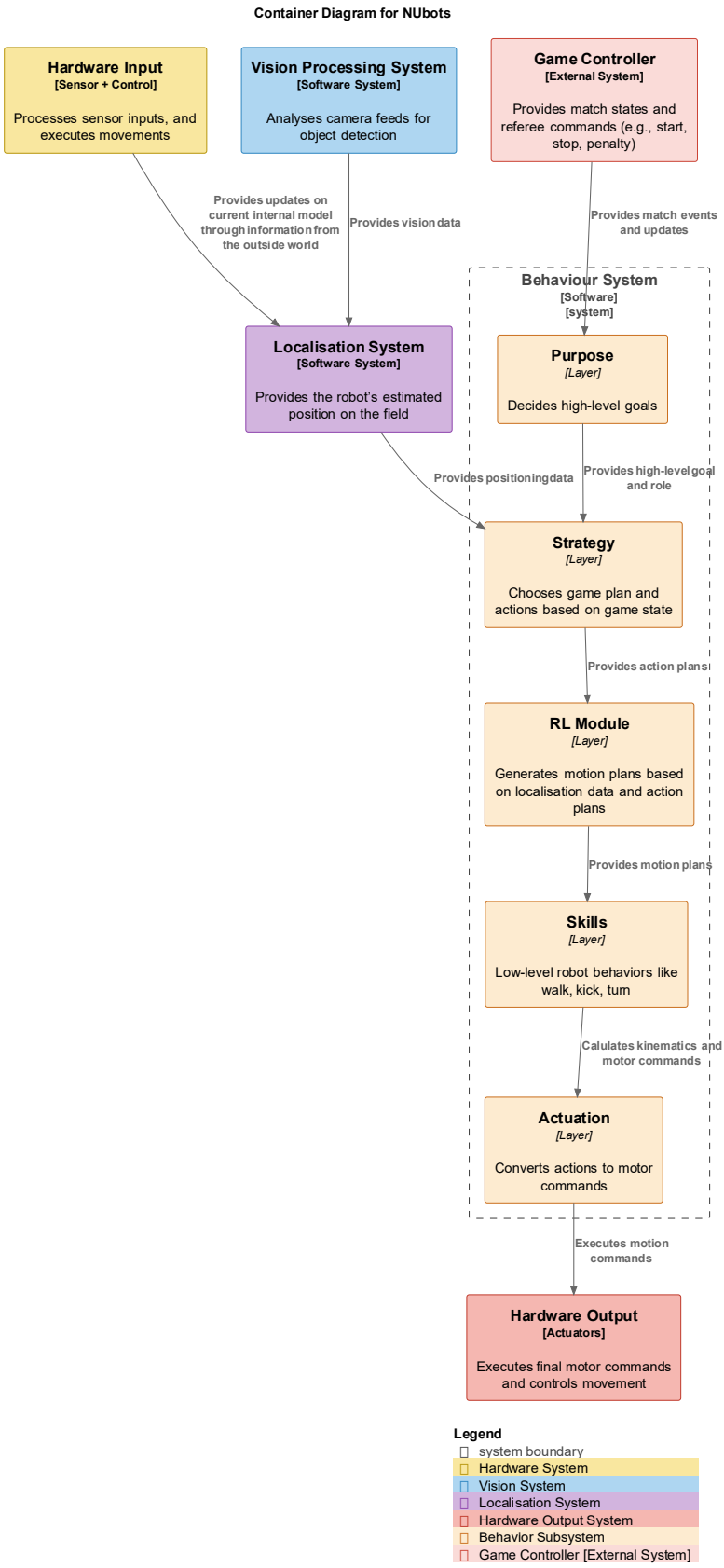

Figure 1: Level 2 C4 Container Diagram showing the RL module (new component) within the Behaviour Container

The RL system represents the current game state using a 12-element array for features like the robot's velocity, position, rotation, ball position, distance, opponent position, goal distance and a Boolean flag for representing whether the robot has ball possession. The trained model outputs a 3-element array for the movements in the x and y directions and the rotation of the robot.

The expected system flow for the system once it is in operation is:

- The Vision and Localisation systems emit messages via the robot's sensors. The simulation environment acts as a substitute for this during training.
- The RL training pipeline trains models on some specified number of timesteps, which provides learning for the model to understand good and bad behaviours.
- The ONNX Conversion pipeline will then be used to convert the trained model into ONNX format for inference (i.e. which is to use the trained model to decide for an action given the current game state, also known as the observation state).
- The NUClear Inference module will be used to load the ONNX model and receive messages from other parts of the NUbots system (for the sensor inputs that SoccerEnv was emulating)
- The predicted action is passed into the Skills Layer through a NUClear message, which then translates them into action commands compatible with the hardware layers.
- The lower layers then execute these actions, translating into the robot's movement.

1.2 Scope

The following items will be included in the testing process:

- **SoccerEnv Simulation Environment:** Validation of the physics simulation, state and observation space representations, reward function performance and action space calculations, as well as updates to the current game state.
- **RL Training Pipeline:** Testing the script used to train models on Proximal Policy Optimisation (PPO) and Deep Deterministic Policy Gradient (DDPG) algorithms.
- **Model Evaluation Framework:** Validation of performance metrics collected such as ball possession time, collisions and falls frequency, goal frequency, etc. success rate calculation, comparison between the RL algorithms and the current rule-based system.
- **ONNX Model Conversion:** Testing the PyTorch to ONNX conversion accuracy to validate the logic implemented to enable conversion, inference output accuracy comparison between the PyTorch and ONNX models (due to the expectation that conversion may introduce some numerical precision differences due to framework-specific implementation of floating-point values), and performance optimisation for the 50Hz frequency requirement.
- **NUClear Inference Module:** Message passing integrations between ONNX models and robots, accuracy of commands outputted by the ONNX models, time gap taken to generate outputs given inputs, sensor data preprocessing.
- **End-to-End System Integration:** Complete workflow testing from sensor input to motor commands, fallback mechanisms, error handling, and system resilience.
- **Physical Robot Hardware Testing**: Testing the models on physical robots for deployment after simulation-based validation, although using physical robots is considered as a stretch goal.

The following items will not be included in the testing process:

- **Sensor Input Validation**: Sensors such as from the vision and localisation systems will not be tested as the project focuses on strategic decision-making rather than perception
- **Model Inference API Integration:** Testing the response from the API call is not considered in-scope as the API is not being designed as part of the project but is instead an externally developed API that NUbots will use.

1.3 Expected Outcome

The testing process will validate that the RL-based system meets all functional and performance requirements while showing significant improvements in performance and results compared to the current rule-based system. Testing will confirm learning convergence between the two RL algorithms with validation for improvements through performance metrics such as ball possession time, fall and collision frequency, decision-making speed, trial success rate, measured via improvement thresholds defined as follows:

- **Ball Possession Time:** ≥15% increase over the rule-based baseline
- **Fall/Collision Frequency:** ≥25% reduction in incidents per game compared to the rule-based system
- **Decision-Making Speed:** Consistent ≤20ms inference latency
- **Trial Success Rate:** ≥60% success rate in simulation tests (100 evaluation trials) with success defined as reaching opponent's goal while maintaining ball possession
- **Goal Achievement Rate:** ≥10% improvement in successful goal approaches under opponent pressure

End-to-end system validation will ensure validation of complete workflows from training through to message passing to the robots. Integration testing will be used to ensure the functionality of message passing from the models to the robots is completed and meets the frequency and latency requirements as specified, as well as the successful conversion between PyTorch to ONNX models for inference. Unit testing will validate the physics of the simulation environment, reward function outputs, state and action space representations. Evaluation frameworks will ensure transparency and clarity for model validation through graphical visualisations.
The system will demonstrate real-time operation with inference latency under 20ms and seamless integration with the NUbots architecture via the NUClear message passing system. Model usage will be validated through randomisation of environment information such as the robots and ball positions, ensuring that learned behaviours generalise across unseen game situations, including opponent strategies and field conditions.

2. TESTING STRATEGY

2.1 Testing Objectives

Comprehensive testing will be conducted throughout the project to validate the implementation of the defined requirements, to detect and fix as many errors as possible throughout the project's development and validate that the developed solution produces performance improvements over the existing rule-based system. The testing phase aims to:

- Report all exceptions and deviations from expected outcomes, including errors in physics calculations (such as deviation from expected values of the environment state, like ball position after it has been moved by a robot and the direction of its velocity) and errors in training evaluation (such as anomalies in the loss function, gradient explosions, etc.).
- Verify that the solution produces the anticipated result and confirm that the two RL algorithms (PPO and DDPG) achieve convergence within 1.5 million training timesteps defined as:
    - A rolling average reward improvement <5% over 50000 timesteps and policy loss decreasing as time progresses to eventually stabilise within minimal variance in loss values.
    - Outperform the rule-based system by ≥15% ball possession time, ≥25% reduction in collisions, ≥10% improvement in goal approach success rate (measured across 100 simulation episodes), ≥60% success rate in simulation tests (100 evaluation trials) with success defined as maintaining ball possession and reaching the opponent's goal and deliver consistent decision making with ≤20ms inference latency.
- Ensure 100% of critical and major defects identified during testing are documented with plans for resolution.

- Ensure that existing systems/processes are not adversely affected: Skills layer, Motion module, and Vision system maintain baseline performance levels (i.e. any non-enhanced functionality remains working).
- Establish the reliability of the RL-based system over the existing rule-based system. This can be measured via Mean Time Between Failures (MTBF) during continuous operation to validate ≥80% system uptime during 8-hour continuous training sessions and ≥99.9% inference availability during match simulation, measured through automated monitoring of system crashes, training interruptions, and inference timeouts.
- Validate the accuracy of the physics and whether realism is sufficient (defined by correlation between simulation and outputs of the physical robot) in the SoccerEnv simulation environment for transferring the models to be used with physical robots:
    - Physics accuracy can be validated by checking that ball trajectory calculations are within ±2% of real-world physics measurements
    - Collision detection to be validated with ≥95% accuracy compared to physical robot collision sensors
    - Validating motor response in simulation through checking that the latency simulation is within ±10ms of actual robot response times
- Ensure the seamless communication between Python training components and NUClear Inference through ONNX conversion within ±10^-4 numerical tolerance.

2.2 Type of Tests to be Undertaken

- ☒ Unit & Component testing
- ☒ System Testing
- ☐ User Acceptance Testing
- ☒ Integration Testing
- ☒ Performance Testing

Unit & Component testing: Necessary for verifying state space, observation space, reward function, neural network architectures, validating the training loops implemented to ensure the trained models are being trained and in ensuring the correctness of math and physics used for the implementation of environmental dynamics through techniques such as Equivalence Partitioning and Boundary-Value Analysis to confirm the accuracy of the physics and math involved in the simulation environment such as the requirements for achieving ball trajectory calculations with ±2% accuracy compared to real-world physics, collision detection with ≥95% accuracy compared to physical robot collision sensors and validating that the response times of the robots within the *SoccerEnv* simulation environment are within ±10ms of the physical robot's response times. Unit testing is critical for objectives related to ensuring the mathematical accuracy of training algorithms and the RL training pipeline by validating individual component functionality before system integration.

System testing: Required for end-to-end validation of the complete RL training pipeline from training through to deployment. System testing is critical for validating the integration between the simulation environment, training systems, model conversion, and inference modules operating within the NUbots systems. System testing supports the objectives of developing an RL-strategy for dribbling behaviours and is required to validate that the performance of the trained models and the system is an improvement over the existing rule-based system in multiple aspects including ≥15% increase in ball possession time, ≥25% reduction in collisions, ≥10% improvement in goal approach success rate (measured across 100 simulation episodes), ≥60% success rate in simulation tests (100 evaluation trials) with success defined as maintaining ball possession while reaching the opponent's goal and deliver consistent decision making with ≤20ms inference latency tested on physical robots for real RoboCup tournaments.

Integration Testing: Critical for testing interfaces between Python training components and C++ deployment systems, validating ONNX model conversion accuracy, NUClear message-passing integration, and cross-platform compatibility between development, testing (due to the simulation

environment being used in the testing environment) and deployment environments. This is integral for the objectives related to integrating the RL module with the NUbots stack by ensuring seamless Python-to-C++ pipeline operation and validation that communication occurs between the RL module and the lower layers of the system at 50Hz frequency and validate that the RL agent outputs actions to the Skills layer within 300ms of gaining a new game state in response to the opponent's behaviour and changing game state. This will also include validation that NUClear message-passing from the RL module will have ≥95% transmission success rate across test transmissions simulating a real match's load for model inference.

Performance Testing: Essential for validating real-time constraints including 50Hz decision frequency, 20ms inference latency, and memory usage optimisation. This phase of testing is important for supporting the objectives related to optimal dribbling with communication rates at 50Hz by verifying real-time performance constraints and supporting deployment readiness for competitive RoboCup environments. Performance testing ensures the system meets operational requirements for stretch goals such as testing RL models on physical robots to simulate the real load on the system during RoboCup games, such as having ≥80% uptime during continuous training and simulation runs across an 8-hour window.

User Acceptance Testing: Not required as the RL-based system is a research component that will be used in the backend without end-user interfaces once it has been deployed. The project focuses on algorithmic performance rather than user experience, with validation conducted through performance metrics and research results rather than user feedback, since the RL module is meant to improve the performance of the robots in RoboCup competitions.

2.3 Assumptions and Constraints

Assumptions:

- PyTorch and Stable-Baselines3 provide reliable and verified RL algorithm implementations rather than developing them from scratch
- 2-D simulation environment provides enough realism and complexity required for meaningful policy training required while maintaining efficiency in training models
- Testing environment's hardware specifications remain stable throughout the testing period

Resource Constraints:

- Testing may be limited to simulation environments due to restricted access to physical NUbots hardware during the development phase, in case of any repairs due to the RoboCup competitions
- Focus on single agent testing due to complexity limitations, with multi-agent coordination deferred for future work to be completed in a separate project by future NUbots members

Project Constraints:

- Project has a deadline to be completed by 26th September 2025 which constrains the overall testing schedule
- Model complexity is limited by the project scope which focuses on dribbling behaviour. Out of scope behaviours include getting up from a fall, defending the goal, etc., which are not incorporated in training
- Integration testing scope limited to essential modules and components required for basic system operation which include the integration between PyTorch to ONNX model conversion, loading and execution of ONNX models in the NUClear environment, message passing between NUClear and the Skills Layer.

2.4 Entry and Exit Criteria

Testing Phase	Entry Criteria	Exit Criteria
Unit Testing	• Source code for the module is complete and reviewed • Required dependencies (PyTorch, Stable-Baselines3, ONNX Runtime. Gymnasium) must be installed and verified • SoccerEnv environment initialises successfully without errors • RL algorithms from Stable-Baselines3 instantiate without any errors	• 100% of severity 1 and 2 defects unit test cases executed and passed • ≥90% of severity 3 defects resolved with the remaining issues documented with impact assessment and scheduled for a future iteration of testing to be conducted by another NUbots member if the project reaches the deadline • ≥70% of severity 4 resolved, remaining low-priority issues logged in backlog for future consideration as learning tasks for newer NUbots members • Equivalence class partitioning tests show correct behaviour across all equivalence classes for all physics calculations, including position, velocities, stored in the observation and action space vectors. • Boundary-value analysis tests show correct behaviour across all possible inputs. • Edge case scenarios are handled without failures or crashes • No memory leaks or issues with simulations such as cascading NaN errors or reward explosions during test executions. Non-critical errors such as warning messages, minor logging issues, cosmetic issues will be documented and approved for deferral if core functionality is not affected.
System Testing	• All requirements are mapped to test cases • Test environment configured to mirror real game conditions	• All high-priority test cases executed and passed • All severity 1 defects resolved or approved for

Testing Phase	Entry Criteria	Exit Criteria
	• Complete training pipeline executes end-to-end without failures • Model evaluation framework produces consistent results meeting the success criteria across multiple runs • Source code for the NUClear Inference module is complete and reviewed by a senior NUbots member, and all integration tests pass successfully • ONNX Conversion Pipeline is complete and reviewed • Model Inference produces consistent results across multiple runs • Test environments (easy, medium and hard) are configured and validated	deferral to be mitigated in a future iteration • System performance meets the baseline benchmarks set in the success criteria to outperform the rule-based system by ≥15% increase in ball possession time, ≥25% reduction in collisions, ≥10% improvement in goal approach success rate (measured across 100 simulation episodes), ≥60% success rate in simulation tests (100 evaluation trials) with success defined as maintaining ball possession and reaching the opponent's goal and deliver consistent decision making with ≤20ms inference latency • End-to-end workflow completes consistently across all difficulty levels • Error-handling mechanisms successfully handle failure scenarios
Integration Testing	• All modules involved pass unit testing • Interface specifications are documented • ONNX conversion pipeline successfully exports Stable-Baselines3 models to ONNX format • NUClear Inference module compiles with ONNX Runtime dependencies • NUClear framework integration compiles and builds without errors	• All integration test cases executed and passed successfully • All interface-related defects resolved • Data exchange and transformation verified across modules • ONNX model conversion outputs bit-identical inference results compared to the Stable-Baselines3 model prior to exporting • NUClear Inference module operates with the 50Hz frequency within the latency requirements • Message-passing operates correctly across 60,000 test transmissions (calculated based on 50Hz message frequency × (60 seconds/minute × 20 minutes) average match duration = 60,000

Testing Phase	Entry Criteria	Exit Criteria
		messages per match), representing full RoboCup match operational requirements with ≥95% transmission success rate • Cross-platform compatibility validated on Windows, Linux and macOS
Regression Testing	• Modifications have been made to core components among training code, simulation environment or ONNX conversion pipeline • Previous version performance metrics have been documented and stored • Automated regression test suite configured and reviewed • Model repository contains previously trained models for comparison • Critical functionality test cases have been identified and configured based on system architecture	• All critical path functionality validated against performance metrics • Outputs from the model after ONNX conversion stays within the 10^-4 difference range • Performance metrics maintain or improve baseline performance with zero tolerance for degradation in critical metrics (success rate, inference speed, decision frequency). • Integration interfaces (i.e. Python↔ONNX↔C++) remain compatible without breaking changes • In case of performance regressions, these are documented with approved mitigation strategies
Performance Testing	• System testing demonstrates functional correctness • Testing environment configured to simulate the hardware resource requirements and allocations that would be required in a real RoboCup game • Test cases are configured to measure latency, frequency and memory usage according to requirements • Performance metrics have been selected for comparison	• Decision frequency exceeds 50Hz under normal operating conditions as per requirements • Inference latency remains below 20ms for 95% decisions • Memory usage stays within 2000MB during extended operation to ensure compatibility with other NUbots' concurrent process memory requirements (prevent starving other processes with memory requirements) • System uptime is confirmed to be at least 80% availability during continuous training and simulation runs across an 8-hour window • NUClear message-passing from the RL module is confirmed to achieve ≥95%

Testing Phase	Entry Criteria	Exit Criteria
		transmission success rate across 60,000 test transmissions for model inference

3. ENVIRONMENT SPECIFICATIONS

NUbots has provided three distinct environments for the purpose of enabling comprehensive validation and verification from development through to deployment. Each environment is used for different testing purposes and has different setup responsibilities within NUbots.

3.1 Development Environment

Description: Individual developer computers configured for building and compiling the NUbots codebase, along with training Stable-Baselines3 models and converting them to ONNX formats for inference. These will have the required testing frameworks to conduct unit, system, integration and performance testing.

Minimum Hardware Requirements:

- **CPU**: Multi-core processor (Intel i5/i7 or AMD equivalent) for parallel training
- **Memory**: Minimum 16.0 GB RAM for the requirement of training large neural networks (32.0 GB recommended to run NUbots system seamlessly builds along with the RL-based system)
- **Storage**: 50 GB free disk space for cloning and building the NUbots codebase, model checkpoints and training logs
- **GPU Requirements**: Minimum 1 CUDA-compatible GPU with a 4GB+ VRAM for accelerated training
- **Graphics Card**: NVIDIA GeForce GTX 1650 Ti (or equivalent)

Minimum Software Requirements:

- **Operating System**: Windows 10+, macOS 12+, Linux via Ubuntu 20.04+ LTS
- **Python Dependencies**: Version 3.12+ compatible with PyTorch version 2.7.1 or higher, Gymnasium version 1.2.0 or higher, Stable-Baselines3 version 2.7.0 or higher, Pygame version 2.6.1 or higher
- **C++ Compiler**: Version C++11 or higher
- **IDE or Text Editors**: VSCode or PyCharm (or equivalent)
- **Version Control**: Git client (or with a GUI like GitHub Desktop, etc.) for repository access and branch management
- **Dependencies:** ONNX Runtime 1.22.0 or higher, CMake 3.16+ for build management
- **SoccerEnv Simulation Environment:** Environment configured for training models on RL algorithms through various scenarios for better learning

Responsibility: Individual NUbots developers are responsible for configuring and managing their own local development environments without introducing any deprecated or new components that do not have backwards compatibility for older software components.

3.2 Testing Environment

Description: Testing environment for executing complex testing that requires specific, defined workflows before code can be pushed to production through unit, system, performance, integration testing and cross-platform validation. Physical robots are used in practice games in conjunction with the testing workflows.

Minimum PC Requirements:

- **Operating System:** Linux via Ubuntu 20.04+ LTS

- **Python**: Version 3.12+ compatible with PyTorch version 2.7.1 or higher, Gymnasium version 1.2.0 or higher, Stable-Baselines3 version 2.7.0 or higher
- **C++ Compiler**: Version C++11 or higher
- **PyTest Testing framework**: Version 8.0.x or higher
- **SoccerEnv Simulation Environment:** Sim environment instances configured for deterministic testing using fixed random seeds for reproducible behaviour
- **Command Line Access**: Terminal or Command Prompt access for CLI commands
- **Test Data Management**: Support for isolated test environments with parallel execution of training and evaluation scenarios
- **Continuous Integration Pipeline**: GitHub Actions to ensure automated testing on code commits
- **Continuous Development Pipeline:** GitHub Actions to ensure automated deployments once test cases pass and code is acceptable
- **Standardised Evaluation Metrics:** Collection of metrics across all testing scenarios
- **NUClear Framework**: Message passing architecture operating at 50Hz frequency
- **IDE or Text Editors**: VSCode or PyCharm (or equivalent)
- **Containerisation**: Docker version 28.0.4 or higher
- **Dependencies:** ONNX Runtime 1.22.0 or higher for APIs to be used for model inference, CMake 3.16+ for build management
- **CPU**: Multi-core processor (Intel i5/i7 or AMD equivalent) for real-time inference
- **Memory**: Minimum 16.0 GB RAM for the requirement of training large neural networks (32.0 GB recommended to seamlessly run NUbots builds along with the RL-based system)
- **Storage**: 50 GB free disk space for cloning and building the NUbots codebase, model
- **Compatibility**: RL-based system to be compatible with NUbots software architecture and message protocols

Robot Hardware Specifications:

- **Robot Platform**: NimbRo-OP2X igus Humanoid Open Platform
- **Real-time OS:** Robot operating system optimised for real-time motor control
- **Onboard Computing:** ARM-based processor running NUbots firmware
- **Memory:** 4GB DDR4 onboard memory
- **Communication:** Ethernet/WiFi for NUClear message-passing
- **Sensors:** Camera, IMU, force sensors for game state input
- **Servos:** Dynamixel MX64AR servos for the arms, head of the NUgus robot (two in each shoulder, one in each elbow, and two in the neck). Dynamixel XH540-W270-R and Dynamixel MX106 servos for the legs (two X series in each ankle, one in each knee, and two in each hip, with the two hip yaw servos remaining as MX-106).

Responsibility: The NUbots developers collectively have the responsibility to maintain testing environment configurations defined and stored on GitHub along with standardised tests. Specialised team members are responsible to work on cross-platform testing for different operating systems as well as ensuring compatibility with the physical robots.

3.3 Production Environment

Description: The finalised environment for games involving the real robots such as for practice games, RoboCup games, etc., representing a real-world application of the system, which will be utilised by the NUbots team in preparation and during tournaments. The production environment differs from the training environment such that inference is expected to happen in production, but training may not be required.

Minimum Requirements:

- **Operating System:** Linux via Ubuntu 20.04+ LTS
- **Python**: Version 3.12+ compatible with PyTorch version 2.7.1 or higher, Gymnasium version 1.2.0 or higher, Stable-Baselines3 version 2.7.0 or higher
- **C++ Compiler**: Version C++11 or higher
- **NUClear Framework**: Message passing architecture operating at 50Hz frequency

- **IDE or Text Editors**: VSCode or PyCharm (or equivalent)
- **Containerisation**: Docker version 28.0.4 or higher
- **Dependencies:** ONNX Runtime 1.22.0 or higher for APIs to be used for model inference, CMake 3.16+ for build management
- **CPU**: Multi-core processor (Intel i5/i7 or AMD equivalent) for real-time inference
- **Memory**: Minimum 16.0 GB RAM for the requirement of running large neural networks concurrently (32.0 GB recommended to seamlessly run NUbots builds along with the RL-based system)
- **Storage**: 50 GB free disk space for cloning and building the NUbots codebase, model
- **Compatibility**: RL-based system to be compatible with NUbots software architecture and message protocols

Robot Hardware Specifications:

- **Robot Platform**: NimbRo-OP2X igus Humanoid Open Platform
- **Real-time OS:** Robot operating system optimised for real-time motor control
- **Onboard Processor:** ARM-based processor running NUbots firmware
- **Memory:** 4GB DDR4 onboard memory
- **Communication:** Ethernet/WiFi for NUClear message-passing
- **Sensors:** Camera, IMU, force sensors for game state input
- **Servos:** Dynamixel MX64AR servos for the arms, head of the NUgus robot (two in each shoulder, one in each elbow, and two in the neck). Dynamixel XH540-W270-R and Dynamixel MX106 servos for the legs (two X series in each ankle, one in each knee, and two in each hip, with the two hip yaw servos remaining as MX-106).

Responsibility: The NUbots team validates the system to be production-ready through CI/CD pipelines, and end-to-end system testing through practice games using the physical robots in a RoboCup-like field for deployment and validation.

4. REQUIREMENTS TRACEABILITY MATRIX

ID	Requirement Description	Test Case(s) Description
FR-01	When the robot has possession of the ball, the RL agent shall provide actions to the Skills layer that allow the robot to maintain ball possession, within 300ms of gaining a new game state in response to opponent behaviour and changing game state.	Initialise a scenario within the simulation environment, initialising the robot and the ball's positions to be in proximity to each other and the opponent's position randomised, excluding the ball's position. Measure the time taken between each game state change.
FR-02	The RL agent shall output motion plans to the Skills layer at a frequency of 50Hz (i.e., 50 updated planning commands per second).	Use a dummy input vector and use the *model.predict()* function to get an action from the model and measure the time it took between giving the model an input and when it generated an output. Calculate frequency as the inverse of the time measured and compare with the target frequency.
FR-03	The system shall perform a complete simulation, using a rectangular field with dimensions 9.0 x 6.0 meters, with the centre of the pitch as the centre, which is defined as complete when reaching the opponent's goal post.	Initialise the environment with the kid-size field configuration and validate the observation space (12D) and action space (3D). Test the *reset()* function on whether it re-initialises state variables and the *step()* function on whether it changes the observation space according to the new action taken.
FR-04	The system shall terminate an episode when a goal is scored, or the ball remains stationary for five seconds or more.	Test goal detection with the ball in the target goal area.

ID	Requirement Description	Test Case(s) Description
		Test the simulation timeout with a stationary ball for 5 seconds or more. Validate that the episodes are truncated when the number of steps exceeds the defined maximum number of steps in an episode.
FR-05	The system shall record performance metrics, including ball possession time, completed manoeuvres, and movement efficiency, after each episode and store this data for performance reporting in a structured CSV format with timestamps to be shared through OneDrive.	Verify automated report generation after training sessions. Validate the CSV format and data integrity. Test metric collection such as tracking possession time and calculating movement efficiency.
FR-06	When the opponent robot is detected within 1.5 meters directly in front of the robot, the RL agent shall decide to change direction to maintain ball possession.	Initialise a scenario within the simulation environment, with the robot and the opponent robot's positions to be within 1.5 meters of proximity to each other. Test both the robot's and the ball's change in direction in movement, and validate that the action space outputs reflect decisions based on proximity.
FR-07	The system shall regulate the speed of the robot during dribbling based on opponent proximity, slowing down for precision when opponents are more than 1.5 meters away and accelerating when evasive action is needed when the opponent robot is less than or equal to 1.5 meters away.	Test the robot's change in speed at varying distances from the opponent robot and validate that action space outputs reflect decisions based on proximity.
FR-08	The RL agent shall output motion plans at 50Hz frequency with <20ms inference latency.	Measure the model inference (action generated via *model.predict()*) and the NUClear message output execution times across 100 inferences, validate the frequency is 50Hz operating under load, and that the latency is under 20ms.
FR-09	The NUClear inference module shall accept 12D state vectors and output 3D action vectors within the [-1.0, 1.0] range representing velocity in x,y directions and the rotation angle for the robot.	Test the ONNX model inference with valid state inputs and validate the action range clipping and output format to ensure the values in the action space are within the range [-1.0, 1.0].
FR-10	The ONNX conversion pipeline shall maintain numerical accuracy within 10^-4 tolerance compared to PyTorch models.	Compare PyTorch VS ONNX inference on identical inputs to ensure the difference is within the tolerance value of 10^-4. Test the *DeterministicPPOWrapper* accuracy and perform cross-platform validation.
FR-11	The system shall maintain at least 80% uptime availability during continuous training and simulation runs across an 8-hour window.	Execute the training pipeline for 8 hours training sessions continuously and monitor for any memory leaks and system crashes.
FR-12	The system should log, capture and report all dribbling failures and internal errors within 1 second of occurrence for debugging and diagnostics. The error log must maintain at least the 100 most recent errors with full context data.	Test error recording during training failures. Validate the completeness of the error recorded and the CSV export functionality. After 100+ errors have occurred, validate that the 100 most recent errors have been recorded with their full context data.

ID	Requirement Description	Test Case(s) Description
FR-13	The system shall capture and store the robot's state space data from the external APIs and the simulation environment, at a frequency of 20Hz during all training and evaluation sessions.	Measure the frequency of state capture during training and the deployment frequency with the NUClear integrations.
FR-14	The system should generate visualisations of model performance, including reward trends, success rate over time, and decision distribution for analytical purposes.	Validate that the implemented callbacks record and generate graphs for reward trends during training, success rates over time, etc.
FR-15	The RL agent shall successfully reach the opponent's goal area without losing ball possession in ≥60% of simulation episodes (60% success rate).	Execute 100 episode test runs in *SoccerEnv* with randomised starting positions. Measure success rate defined as reaching the goal boundary while maintaining ball contact throughout the episode to validate that it is ≥60%.
FR-16	When the opponent robot is detected within 1.5 meters of proximity to the robot, the RL agent shall output actions to avoid collisions with the opponent robot.	Test collision detection and avoidance with varying opponent proximity scenarios. Measure collision frequency and validate that evasive actions are generated when the opponent robot is within 1.5 meters of the robot.
FR-17	The RL system shall achieve ≥15% increase in ball possession time over the rule-based system's baseline.	Execute 100 simulation episodes comparing RL vs rule-based performance. Measure average possession time per episode and calculate the improvement percentage for validation that it is ≥15%.
FR-18	The RL system shall reduce fall/collision frequency by ≥25% compared to the rule-based system.	Monitor collision events across 100 episodes with opponent interactions. Compare incident rates between RL and rule-based approaches to calculate the difference in fall frequency to validate fall frequency reduced by ≥25%.
FR-19	The RL system shall demonstrate ≥10% improvement in goal achievement rate under opponent pressure compared to the existing rule-based system's baseline.	Test goal approach success with active opponent interference across 100 episodes. Compare the success rates against the rule-based baseline performance to validate ≥10% improvement.
FR-20	The *SoccerEnv* simulation environment shall achieve ball trajectory calculations with ±2% accuracy compared to real-world physics.	Compare the simulated ball physics against validated and known physics calculations for trajectory, velocity, and collision outcomes.
FR-21	The RL NUClear module shall achieve ≥99.9% inference availability during match simulations.	Monitor the inference module's uptime during simulation sessions, measuring failure rate and recovery time.
FR-22	NUClear message-passing from the RL module shall achieve ≥95% transmission success rate across 60,000 test transmissions for model inference.	Execute 60,000 message transmissions (representing a 20-minute match duration at 50Hz calculated as 50 messages/second x (60 seconds/minute x 20 minutes) = 60,000 messages) simulating worst-case continuous operation scenarios expected during RoboCup tournament conditions. Monitor and measure transmission success and failure rates.

ID	Requirement Description	Test Case(s) Description
FR-23	The RL NUClear module shall achieve 95% inference decisions within a 20ms latency constraint.	Execute 60,000 inference operations to simulate a real match. Measure latency distribution and validate 95% messages fall within the 20ms latency constraint.
FR-24	The RL system shall maintain memory usage ≤2000MB during extended operation concurrent with working with other systems in NUbots, such as Vision and Localisation systems.	Monitor memory consumption during continuous operation sessions. Validate compatibility with concurrent NUbots processes.
FR-25	The *SoccerEnv* simulation environment shall achieve collision detection with ≥95% accuracy compared to physical robot collision sensors.	Execute collision detection scenarios in *SoccerEnv* with known collision events. Compare the detected collisions against the reference collision events and validate whether the accuracy is ≥95% in comparison.
FR-26	The response times of the robots within the *SoccerEnv* simulation environment shall be within ±10ms of the physical robot's response times.	Measure robot response latency in *SoccerEnv* for identical motor commands. Compare the measured latency against the documented NUbots hardware response times and validate the ±10ms tolerance constraint.

5. REGRESSION TESTING

5.1 Regression Testing Planning and Execution

Regression testing will be executed automatically upon significant changes to core system components to ensure new features do not break existing functionality. The regression suite includes critical path testing covering end-to-end workflows, performance benchmarks, and integration stability tests. This will be done via GitHub Actions CI/CD pipeline, triggered automatically whenever code is committed to the branch. This will provide guardrails to guide development to be of high standards, rather than a gate at the end, which could halt development with no transparency into the issues.

Regression testing is triggered when either of the following occurs:

- Changes to the RL Training Pipeline for PPO and DDPG algorithms, such as the number of timesteps, hyperparameters, neural network architecture and other conditions of training
- Modifications to SoccerEnv environment dynamics or reward functions to encourage any new behaviours for the agent
- ONNX conversion pipeline updates or C++ inference module changes
- NUClear integration modifications affecting message-passing or sensor processing
- At the end of a development sprint
- Regression checks before merging feature branches into the main branch
- Complete regression test execution before deployment of the system

Regression testing will be conducted in a separate testing environment with fixed random seeds to provide reproducible environments, standard hardware configurations used in RoboCup games, and consistent hyperparameter settings.

5.2 Integration with Other Types of Testing

Unit Testing Integrations

Unit test cases for key RL components will be included in the regression testing plan:

- **SoccerEnv Environment**: Unit tests (which include Equivalence class partitioning and Boundary Value Analysis) will be required for state space calculations, reward function outputs and physics simulation dynamics

- **ONNX Conversion**: Unit tests will be required to validate the functionality of the deterministic wrapper implemented specifically for the PPO algorithm, tensor shape validation and accuracy of outputs from the ONNX models.

Integration Testing Integrations
Integration test cases that validate the stability between all interfaces that interact with each other as the system is operating will be included in the regression testing plan:

- **Python-to-ONNX Pipeline**: The test cases that will be integrated for regression testing will include validating that the accuracy of the newly converted ONNX models' outputs will be within 10^-4 tolerance compared to the outputs of the PyTorch models
- **ONNX-to-NUClear Interface**: Test cases will be required to test the message format and consistency of the timing of the inference results.
- **NUClear Message Passing**: Test cases will be required to validate compliance with the standardised messaging protocol and preservation of the frequency requirements of 50Hz.

System Testing Integrations
System test cases that validate key workflows of the entire system in operation and performance aspects will be included in the regression testing plan:

- **Integrity of the RL Training Pipeline**: Test cases that validate the successful running of complete training cycles with validation that there is convergence to desirable behaviour based on performance metrics will be included in the regression testing plan.
- **Model Evaluation**: Analysis of the model's performance observed through performance metrics will be conducted to validate that they are within the acceptable ranges. Validation that the system logs and reports all internal errors for debugging and diagnostics.
- **Complete Inference Workflow Validation**: Testing the complete workflow of a trained model, being given the current state observations and predicting the next action via inference and sending the command through a message in NUClear to the correct modules and validating that the robot received these commands, will be included in the regression testing plan.

Performance Testing Integrations
Performance test cases that validate the high priority "must-haves" requirements will be included in the regression testing plan:

- **Inference Latency**: The requirement to maintain <20ms latency across model updates and system changes will be included.
- **Performance Improvements**: The requirements to achieve a ≥15% improvement in ball possession rate, ≥25% reduction in collision frequency and ≥10% improvement in goal approach, etc. will be included.

5.3 Test Case Reuse and Maintenance Plan

Baseline test cases will be maintained as versioned test cases that will be automatically executed.
Core Regression test cases:

- **FR-01**: Testing that the RL agent outputs action commands to the Skills Layer and measuring the time duration taken between each change in game state is within 300ms.
- **FR-02**: Testing that the RL agent outputs action commands to the Skills Layer at a frequency of 50Hz.
- **FR-03**: Testing SoccerEnv Initialisation and reset functionality
- **FR-06**: Test that the robot and ball (when it has possession of the ball) change direction of movement when the opponent robot is less than 1.5m away from the primary robot and validate that action space outputs reflect decisions based on proximity.
- **FR-07**: Test that the robot's speed changes at varying distances from the opponent robot and validate that action space outputs reflect decisions based on proximity.
- **FR-08**: Measure the model inference (action generated via *model.predict()*) and the NUClear message output execution time across 100 inferences and validate the frequency is 50Hz operating under load, and validate the latency is under 20ms.
- **FR-09**: Test the ONNX model inference with valid state inputs and validate the action range clipping and output format to ensure values in the action space are within the range [-1.0, 1.0].

- **FR-10**: Compare PyTorch VS ONNX inference on identical inputs to ensure the difference is within the tolerance value of 10^-4. Test the *DeterministicPPOWrapper* accuracy and perform cross-platform validation.
- **FR-15**: Execute 100 episode test runs in *SoccerEnv* with randomised starting positions. Measure success rate defined as reaching the goal boundary while maintaining ball contact throughout the episode to validate that it is ≥60%.
- **FR-16**: Test collision detection and avoidance with varying opponent proximity scenarios. Measure collision frequency and validate that evasive actions are generated when the opponent robot is within 1.5 meters of the robot.
- **FR-17**: Execute 100 simulation episodes comparing RL vs rule-based performance. Measure average possession time per episode and calculate the improvement percentage for validation that it is ≥15%.
- **FR-18**: Monitor collision events across 100 episodes with opponent interactions. Compare incident rates between RL and rule-based approaches to calculate the difference in fall frequency to validate fall frequency reduced by ≥25%.
- **FR-19**: Test goal approach success with active opponent interference across 100 episodes. Compare the success rates against the rule-based baseline performance to validate ≥10% improvement.
- **FR-20**: Compare the simulated ball physics against validated and known physics calculations for trajectory, velocity, and collision outcomes
- **FR-22**: Execute 60,000 message transmissions (representing a 20-minute match duration at 50Hz calculated as 50 messages/second x (60 seconds/minute x 20 minutes) = 60,000 messages) simulating worst-case continuous operation scenarios expected during RoboCup tournament conditions. Monitor and measure transmission success and failure rates.
- **FR-23**: Execute 60,000 inference operations to simulate a real match. Measure latency distribution and validate 95% messages fall within the 20ms latency constraint.
- **FR-25**: Execute collision detection scenarios in *SoccerEnv* with known collision events. Compare the detected collisions against the reference collision events and validate whether the accuracy is ≥95% in comparison.
- **FR-26**: Measure robot response latency in *SoccerEnv* for identical motor commands. Compare the measured latency against the documented NUbots hardware response times and validate the ±10ms tolerance constraint.

Test case Adaptation:
When requirements change, test cases will be altered accordingly post systematic review:
- Identify which test cases require modification based on the changed requirements through the Requirements Traceability Matrix for correlation.
- Update performance baselines when improvements require new standards.
- Test case parameters may require changes while the core logic of the test case may remain unchanged.
- Maintain traceability between requirement changes and corresponding test case modifications.

Test Data Management:
Regression test data includes well-defined scenarios, model checkpoints and performance baselines stored in GitHub for version control. Versioning the test data enables comparisons across different sprints and allows for rollbacks to previous states as needed.

5.4 System Areas Most Vulnerable to Regression

Areas most vulnerable to regression include:
- **Training Algorithm Stability**: The RL training pipeline has the highest risk of regression due to the stochastic (random) nature of reinforcement learning (specifically the PPO algorithm which outputs a probability distribution over actions) and interdependent hyperparameters. Changes to the reward function, environment dynamics, neural network architectures or training procedures can change convergence behaviour, which requires validation against performance metrics and

established baselines. The regression testing plan includes using training curves, convergence rates and resulting performance metrics to catch any degradation earlier.

- **SoccerEnv Simulation Environment**: The simulation environment is vulnerable to regression because of the interdependent physics calculations for the environment dynamics, maintaining the latest state space and reward function implementation. Changes to the ball physics (such as mass, velocity, friction of the ball, etc.), collision detection and robot action dynamics can impact the training and behaviour of models. Modifying the observation (12-D vector) or action spaces (values in the [-1.0, 1.0] range) could break existing trained models. The regression testing plan includes the validation of the physics simulations, consistency between the observation and action space during the simulations, and environment reset functionality to ensure that the simulation updates between episodes.
- **ONNX Conversion Pipeline**: The Python-to-ONNX-to-C++ conversion pipeline is vulnerable because it involves multiple framework transitions with potential for numerical precision loss or interface incompatibilities. The *DeterministicPPOWrapper* represents a critical component where changes could break the stochastic-to-deterministic conversion logic, which is essential for model deployment. Regression testing validates conversion accuracy through bit-level comparison and cross-platform inference testing (i.e. comparing outputs from the PyTorch models with the ONNX models).
- **NUClear Integration Interfaces**: Message-passing protocols between the training pipeline and C++ deployment infrastructure have vulnerabilities to regression. Changes to message formats, timing requirements, or communication protocols could break real-time robot operation. The regression testing plan includes validation of protocol compliance, message frequency requirements (50Hz), and interface compatibility across system updates.
- **Performance-Metrics Measurement:** Real-time performance requirements (50Hz decision frequency, <20ms inference latency, etc.) create regression risks where minor changes can cause performance degradation that renders the system unsuitable for robot deployment. Regression testing continuously monitors timing and resource usage to prevent performance regressions that could hinder competition readiness.
- **Management of Configurations:** The field configuration system (which includes field dimensions and physics values used in calculating environment dynamics) affects multiple system components and represents a significant regression risk. Changes to field dimensions, difficulty settings, or reward parameters can have cascading effects across training, evaluation, and deployment systems. The regression testing plan includes validation of configuration consistency and backward compatibility to prevent system-wide failures from configuration changes.

6. DEFECT MANAGEMENT

6.1 Defect Definitions

Severity	Definition
1 (Critical)	The problem results in a complete system outage or functions unavailable/malfunction that seriously impacts testing progress. For example, training pipeline crashes, ONNX conversion failures, and NUClear integration fails causing complete message-passing breakdown.
2 (High)	System functionality is degraded with serious adverse impact to testing, but effective workarounds exist. For example, training convergence failures, inference latency exceeding the defined maximum, and memory leaks during extended operation.
3 (Medium)	System functionality is degraded with a moderate adverse impact to the testing progress, and the error can be rectified by a workaround. For example, suboptimal hyperparameter usage, logging system issues and graph visualisation issues, outdated documentation, incorrect validation operations, etc.
4 (Minor)	There is no immediate adverse impact to testing. For example, typographical errors in graphs or the CLI, spacing issues or colour issues with the simulation environment for the soccer simulation, etc.

Priority	Definition
1 (Immediate)	Resolved within 4 hours of discovery. Typical for Severity 1 and critical Severity 2 defects
2 (High)	Resolved within 8 hours of discovery. Typical for Severity 2 defects or urgent issues with visualising either the training or the soccer simulation
3 (Medium)	Resolved within 24 hours of discovery. Typical for Severity 3 defects or non-blocking issues
4 (Low)	Resolved within 40 hours of discovery. Typical for suggestions for improvements or modifications to the visualisation

6.2 Defect Action Plan

Roles during testing:

- Main Developer: Ali Riyaz
- Tester: Ali Riyaz
- Project Supervisor: Khaled Saleh
- Component Owners or development team: Various NUbots members

Defect identified during test phase	Action
Unit	Defects identified during the unit testing phase will be addressed immediately. Critical defects (Severity 1-2) affecting core RL components, training algorithms, or environment functionality are escalated to the project supervisor for more direction. Non-critical defects are documented and resolved after critical issues are addressed.
Integration	Integration defects are assigned to the component owner responsible for the affected interface. ONNX conversion defects are addressed by the main developer, while NUClear integration issues are escalated to senior NUbots members. Critical defects require immediate cross-team collaboration with NUbots members experienced with NUClear for timeline and resource coordination.
System	System-level defects are tracked by the main developer and addressed with assistance from senior NUbots members. End-to-end workflow failures require a comprehensive root cause analysis involving the main developer and other NUbots members. Performance defects affecting real-time requirements (50Hz frequency, 20ms latency, etc.) receive immediate priority due to deployment criticality.
Performance	Defects are tracked and addressed by the main developer where the performance defect was identified. Latency violations (>20ms for inference) trigger immediate investigation with potential algorithm simplification or hardware requirement reassessment.
Regression	Regression defects indicate potential system instability and receive the highest priority assignment to be addressed by the main developer. Performance regressions require immediate investigation and potential rollback to previous working versions. Integration regressions affecting ONNX conversion or NUClear message-passing trigger full system validation.

6.3 Defect Resolution

Defect Documentation Tools: Defects are logged through Jira with standardised templates to ensure complete information and context are communicated.

Defect Resolution Process:

1) **Defect Detection and Initial Review**: Defects are identified in any phase of testing through first automated execution of test cases. The discoverer reviews the defect to determine the severity and priority of the defect.
2) **Defect Logging**: The defect is logged via Jira with a unique ID for identification, title, defect description, severity and priority classification, affected component, pre- and post-conditions for reproducible steps for context, environment specifications, proof of defect, which includes screenshots or outputs and any dependencies or blockers with other work items.
3) **Defect Assignment:** Defects are reviewed when lodged, and within 4 hours of logging the defect, are assigned to the main developer and may require senior NUbots members depending on the defect. The team is notified according to the severity and priority of the lodged defect. For example, severity 1 and/or priority 1 defects trigger notifications to the developer team for immediate resolution.
4) **Assessment and Resolution:** The main developer investigates the defects and works to resolve them. Once the defect has either been resolved or a new decision has been made regarding the defect, then the main developer must include information such as the root cause of the defect, code fixes with comments, update the associated test cases, and verify that the fix does not cause regression in other components of the system.
5) **Verification and Further Testing:** Once the defect has been resolved, the main developer is assigned to verify that the fix has resolved the defect and conduct additional regression testing on related components. Verification requires reproducing the original steps that discovered the defect and additional path testing for different scenarios to verify that no regression defects have been produced.
6) **Updated Documentation**: After the defect has been verified to be fixed, it is marked as done in Jira. In case of any defects with statuses incomplete or in discovery, additional comments about the decisions related to that defect are included for future consideration. Documentation in NUbook will be updated to include learnings to prevent the defect from reappearing in other work items.

Defect Reporting Standard:
Defects are discussed during separate recurring meetings to track the status of defects and further comments for transparency into the issues. All tracking is done via Jira tickets which are used for information related to defects, along with resolution and links to the commits that include the fix and the associated documentation.

6.4 Testing Suspension and Resumption Criteria

Testing will be suspended when any of the following occur:

- **System Outages and Failures**: Critical system failures from severity 1 defects in components such as the RL training pipeline, ONNX Conversion pipeline, NUClear Inference module require immediate resolution, which warrants halting testing activities.
- **Integration Breakdown**: Python-to-C++ interface failures, message-passing breakdowns that prevent integration testing and system testing.
- **Training Cascade Failures**: Errors that cascade during training, such as NaN or memory leaks affect the model's behaviour, which must be resolved immediately.
- **Simulation Environment Crashes**: *SoccerEnv* is utilised as the environment to train models. In case of any defects within this component, training gets compromised, which calls for immediate resolution of the defect before resuming testing.
- **Data Corruption**: When models, training data and configuration files necessary for operating the system get corrupted.
- **Version Control Issues**: Issues with rollbacks or versioning the current system state call for immediate resource allocation to remedy such issues.

Resumption can only occur when all the following have occurred:

- **Resolution of Blocking Defects**: When all blocking defects have been investigated, resolved and verified.

- **Training and Inference Stability**: When the system has demonstrated training and model inference outputs reliably without degradations and no critical issues.
- **Performance Metrics Documentation**: When the system is confirmed to document and collect performance metrics accurately and reliably for model comparison.
- **Python-ONNX-NUClear Integration Verification**: When all interfaces are confirmed to be operating correctly with reliable message-passing and data exchange operations.
- **Documentation Updates**: When all documentation associated with the defects has been updated along with their resolution, status and prevention measures.