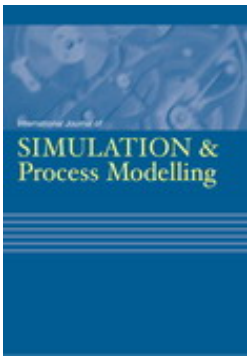
INDERSCIENCE PUBLISHERS
Linking academia, business and industry through research



International Journal of Simulation and Process Modelling

ISSN online: 1740-2131 - ISSN print: 1740-2123
https://www.inderscience.com/ijspm

Automated simulation testing for complex software environments using multi-agent reinforcement learning

Qiuming Zhang, Jing Luo

DOI: 10.1504/IJSPM.2025.10075279

Article History:

Received:	14 October 2025
Last revised:	16 November 2025
Accepted:	18 November 2025
Published online:	06 March 2026

Copyright © 2026 Inderscience Enterprises Ltd.

Automated simulation testing for complex software environments using multi-agent reinforcement learning

Qiuming Zhang

School of Computer Science,
China University of Geosciences,
Wuhan, 430074, China
Email: qmzhang@cug.edu.cn

Jing Luo*

Library and Archives,
Wuhan Vocational College of Software and Engineering,
Wuhan, 430205, China
Email: 53200121@whvcse.edu.cn
*Corresponding author

Abstract: The growing complexity of software systems poses major challenges for automated testing in continuous integration and continuous delivery pipelines. This paper proposes multi-agent-based software automated testing, a multi-agent reinforcement learning framework that models testing tasks as a decentralised partially observable Markov decision process. Using the Q-network mixing algorithm, the system enables coordinated decisions across testing agents. Evaluation on a TravisTorrent-based simulation environment shows multi-agent-based software automated testing achieves a 95.2% defect detection rate – showing a 4.7% improvement over the best multi-agent baseline – while reducing test execution time to 70% of conventional rule-based scheduling. Compared with multi-agent deep deterministic policy gradient, it demonstrates a large effect size (Cohen's d = 0.89) in defect detection. These results demonstrate the framework's effectiveness in improving testing quality and efficiency, offering a viable solution for intelligent test automation in complex software environments.

Keywords: multi-agent reinforcement learning; MARL; automated testing; continuous integration and continuous delivery; CI/CD; Jenkins.

Reference to this paper should be made as follows: Zhang, Q. and Luo, J. (2026) 'Automated simulation testing for complex software environments using multi-agent reinforcement learning', *Int. J. Simulation and Process Modelling*, Vol. 23, No. 1, pp.1–10.

Biographical notes: Qiuming Zhang is an Associate Professor in the School of Computer Science at China University of Geosciences, China. He received his PhD from China University of Geosciences in 2009. His research interests include intelligent computing and machine learning.

Jing Luo is a Senior Engineer in Library and Archives at Wuhan Vocational College of Software and Engineering, China. She received her Master's degree from China University of Geosciences in 2009. Her research interests include intelligent education and machine learning.

1 Introduction

With the deepening of digital transformation, software has become the core infrastructure to support the operation of modern society. In this context, the ability to quickly respond to market demands and continuously deliver high-quality software products has become the core competitiveness of an enterprise, and the DevOps culture and its key technology practice, continuous integration and continuous delivery (CI/CD), have significantly shortened the software release cycle through the automation of the build-test-deploy pipeline, laying the engineering cornerstone of modern agile development (Humble and Farley, 2010; Faustino et al., 2022). As the quality guardian of the CI/CD process, the effectiveness of the automated testing system directly determines the trustworthiness of the delivered software. However, as the complexity of software systems themselves grows exponentially, their operating environments exhibit a high degree of dynamism, heterogeneity, and uncertainty, which poses unprecedented and serious challenges to traditional automated testing approaches (Dragoni et al., 2017). Classical automation testing strategies, whether based on fixed-order test case execution or relying on manually predefined heuristic rules,

Copyright © The Author(s) 2025. Published by Inderscience Publishers Ltd. This is an Open Access Article distributed under the CC BY license. (http://creativecommons.org/licenses/by/4.0/)

often fail to meet expectations in terms of test coverage, defect detection efficiency, and resource utilisation efficiency when facing such complex environments full of uncertainties (Mukherjee and Patnaik, 2021).

To address this challenge, academia and industry have begun exploring the integration of AI techniques into the software testing process. In recent years, reinforcement learning (RL)-based testing methods have shown impressive potential. Unlike traditional approaches, RL empowers test agents to learn and make decisions autonomously through continuous interaction with the environment, enabling them to dynamically adapt to changes in software state (Sutton and Barto, 1998). Existing research has successfully applied single-intelligence RL to specific tasks such as test case generation, test prioritisation and fault localisation (Moghadam et al., 2022). For example, some work has utilised deep Q-networks (DQNs) or policy gradient methods to guide single agents to explore in the input space or graphical user interface (UI) of a program to maximise coverage of a specific code structure or to discover potential crashes (Mnih et al., 2015). These seminal studies confirm the feasibility of RL in solving the software test sequence decision problem. However, we observe that most of these approaches are based on the simplified paradigm of 'single intelligence, single task'. In a real CI/CD scenario, testing activities are inherently concurrent, collaborative, and highly intertwined (Shahin et al., 2017): unit testing, integration testing, system testing, and various types of non-functional testing form a complex ecosystem. A single 'omnipotent' intelligence has difficulty in acquiring expertise in all testing tasks at the same time, and its decision space expands dramatically with task complexity, leading to inefficient learning, difficulty in convergence, and inability to effectively characterise the intrinsic dependencies and competitions among testing tasks (Arcuri and Fraser, 2013). This paradigmatic limitation makes it difficult to directly apply existing single-agent-body RL methods to large-scale, distributed, modernised CI/CD pipelines.

The core feature of complex CI/CD environments is their intrinsic multi-agent interaction nature. A typical software build process consists of dozens or even hundreds of interrelated steps, such as code compilation, dependency deployment, database migration, execution of multiple test suites, etc. (Dimitroulakos et al., 2009). These steps are supported by different tool chains, run in heterogeneous computing environments, and compete for shared resources (e.g., compute, storage, network bandwidth). Currently, the execution order and resource allocation of these steps are mostly determined by static configuration files (e.g., Jenkinsfile) or simple dependencies, and lacks the intelligence to dynamically adjust based on the real-time system state (e.g., the characteristics of the current code change, the current server load, and the historical build success rate) (Vasilescu et al., 2015). In this study, we take the example of Jenkins, a widely used automation server in the industry, which is the very core coordinator of the CI/CD process and is responsible for executing a series of tasks defined in the pipeline. The build logs it generates naturally record the complete lifecycle data of the pipeline execution, and thus become a key source of information for us to build simulation environments and learn intelligent decision-making strategies. This rigid scheduling mechanism is highly susceptible to uneven resource utilisation – certain test tasks wait while resources are free, while critical tasks are queued due to lack of resources, ultimately lengthening the execution time of the entire pipeline (Barr et al., 2014). More importantly, it fails to achieve the global optimality of the testing strategy. For example, when the integration test intelligence detects an anomaly in the response of a particular microservice interface, it should be able to collaborate with the unit test intelligence to focus on testing the relevant modules of that service, and at the same time notify the performance test intelligence to adjust the load strategy. Such complex collaboration strategies are far beyond the planning capabilities of existing single-intelligence systems. Thus, modelling the testing activities in a CI/CD pipeline as a multi-agent system and coordinating its behaviour using multi-agent reinforcement learning (MARL) is not only a technologically natural evolution, but also an inevitable choice to address the core pain points in the field (Stone and Veloso, 2000).

Despite the remarkable success of MARL in areas such as robot collaboration and intelligent transportation (Du and Ding, 2021; Zhang et al., 2021), its application in software engineering, especially in automated testing, is still in its infancy. Introducing MARL into CI/CD automated testing is not a simple technology grafting, but involves a series of fundamental challenges and innovations. First, and most critically, is the problem of how to formalise a specific CI/CD test environment into a standard MARL. This requires a precise definition of the intelligentsia (who is making decisions), the observation space (what the intelligentsia can see), the action space (what the intelligentsia can do), and the reward function (how to evaluate the goodness of decisions) (Busoniu et al., 2008). The design of the reward function is particularly critical, as it must be able to balance short-term gains (e.g., quickly detecting an apparent error) with long-term gains (e.g., ensuring system stability through thorough testing) and reconcile potential conflicts between individual agents' goals (e.g., unit tests completed quickly) and global system goals (e.g., overall pipeline is efficient and of high quality) (Rashid et al., 2020). Second, the choice of algorithms must match the structure of the problem. CI/CD environments often require agents to make decisions with only localised access to information, but at the same time their utility is tightly dependent on the overall performance of the team. This requires that the MARL algorithms we employ must be capable of handling 'collaborative' and 'partially observable' scenarios (Lowe et al., 2017). Finally, and generally missing from existing research, is the fidelity and rigor of model evaluation. Most RL testing studies have only been validated in a limited number of benchmark programs (e.g., a specific open-source project) or highly

simplified simulators, and the generalisability of their conclusions to real-world, industrial-grade CI/CD pipelines with large scale and complex interactions is questionable (Hessel et al., 2018). A high-fidelity simulation environment built on large-scale, real-world CI/CD log data is essential for an objective and unbiased assessment of algorithm performance.

Based on the above analysis, this study aims to systematically explore the application of multi-agent RL in automated testing of complex software environments, and is committed to bridging the gap between theoretical models and real-world practices in existing research (Foerster et al., 2016). The core work of this paper is to build a data-driven multi-agent testing framework for CI/CD pipelines. We will provide insights on how to build a highly realistic simulation environment using long-term build logs extracted from real Jenkins projects and, based on this, a rigorous formal definition of the problem of co-scheduling test tasks. By adopting cutting-edge multi-agent body collaboration algorithms, we will train a set of testing agents to learn to dynamically adjust testing strategies through collaborative decision-making under complex resource constraints and task dependencies, and ultimately realise the dual improvement of testing efficiency and software product quality. This research is conducted not only to propose a new testing technique, but also to promote the evolution of software engineering automation to a more intelligent, adaptive and collaborative future paradigm.

2 Related work

2.1 Evolution and limitations of automated testing techniques

The technical evolution of automated testing, as a key aspect of software quality assurance, has always centred on how to detect software defects more efficiently and comprehensively. Early research focused on techniques for automatic generation of test cases, e.g., search algorithms based on code coverage criteria (e.g., genetic algorithms, particle swarm optimisation) were widely explored with a view to automatically generating test inputs that can cover specific paths or branches (Panichella et al., 2015). Such approaches, while successful on specific benchmark programs, depend heavily on the design of the fitness function for their performance and usually lack a global optimisation perspective on the sequence of test case execution. On the other hand, test case prioritisation and selection techniques aim to optimise the allocation of testing resources by sorting or filtering the existing set of test cases, e.g., by giving priority to the execution of test cases with higher fault detection rates (Yoo and Harman, 2012). However, most of these approaches rely on static analysis of historical execution data, and their decision models tend to be offline and fixed. When faced with highly dynamic contexts in continuous integration environments – such as frequent code changes, fluctuating system resources, and complex dependencies among test tasks – these static rule-based or offline optimisation approaches appear to be insufficiently adaptable to achieve real-time, dynamic test scheduling and decision making (Dimitroulakos et al., 2009). Their inability to intelligently adjust the strategy of subsequent test tasks (e.g., skipping certain related tests or initiating deeper diagnostics) based on real-time feedback (e.g., the failure of a particular microservice build) during pipeline execution constitutes their core limitation in modern DevOps practice.

2.2 RL for single intelligentsia applications in software testing

In order to overcome the rigidity problem of traditional automated testing methods, researchers have started to introduce RL to empower the testing process to learn and make decisions on its own. Single-intelligence RL has shown initial potential in this area, and the main idea is to model the testing process as a sequential decision-making problem in which a single intelligent learns the optimal testing strategy by interacting with the environment. For example, in graphical UI testing, researchers have used DQNs to train agents to learn sequences of interactions with the application's UI elements in order to maximise activity coverage or detect application crashes (Moghadam et al., 2022). Similarly, in unit test generation, RL was used to guide agents to explore in the input space of a program to generate test data that triggers anomalous behaviour or covers complex branches (Moghadam et al., 2022). These studies have certainly demonstrated the effectiveness of RL in dealing with the problem of exploring and exploiting trade-offs in testing. However, a prevalent paradigm is to frame the test problem within a 'single-intelligence, single-goal' framework. The limitations of this paradigm are exposed when the testing scenario is expanded to a full CI/CD pipeline with multiple heterogeneous, concurrent test tasks. A single intelligent body needs to learn the expertise of all test tasks simultaneously, and its action space grows exponentially with the number of tasks, which can easily lead to dimensionality catastrophe, making the learning process slow and difficult to converge. More importantly, single-intelligence body models are inherently unable to portray and exploit synergies and competing relationships between test tasks, e.g., the failure of an integration test can provide valuable, focused directions of exploration for multiple unit-testing agents, a distributed collaborative decision-making mechanism that is not possible with single-intelligence body architectures.

2.3 Theory and cross-domain applications of RL with multi-intelligence

MARL provides a natural theoretical framework for solving the distributed decision making and collaboration problems described above. MARL investigates how to enable multiple agents to reach common or individual goals through autonomous learning in a shared environment (Busoniu et al., 2008). Its core challenge is environmental non-smoothness, i.e., from the perspective of a single

intelligent, changes in the environment are not only caused by its own actions, but also stem from parallel learning and decision making by other agents. To address this challenge, a series of advanced algorithms have been proposed. For example, the multi-agent deep deterministic policy gradient (MADDPG) algorithm employs the actor-critic framework of centralised training and distributed execution to stabilise the learning process by providing the critic network with information about the strategies of other agents during the training phase (Lowe et al., 2017). The Q-mixing (QMIX) algorithm, on the other hand, guarantees optimal joint action selection even in distributed execution by constraining the overall value function to be a monotonic mixture of individual agents' value functions (Rashid et al., 2020). These algorithms have achieved breakthrough successes in many complex systems, such as cooperative multi-robot control, smart grid management, and optimisation of large-scale traffic light networks (Zhang et al., 2021). In these applications, the agents significantly improve the overall efficiency and robustness of the system through collaboration. These successful cases strongly demonstrate the great potential of MARL in solving complex system problems with distributed characteristics, local observation and global goal optimisation, and provide a solid theoretical and empirical foundation for its application in software engineering.

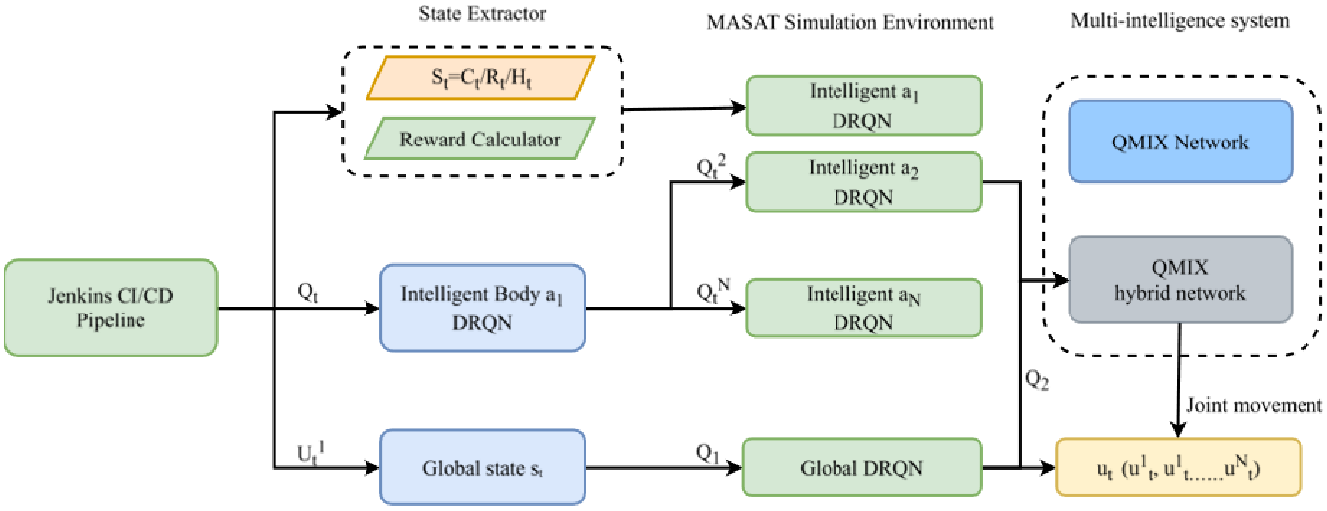
2.4 Summary of research gaps

In summary, existing research presents a clear evolutionary path, but also reveals a critical research gap. While automated testing techniques have laid the foundation for quality assurance, single-agent-object RL has injected adaptivity into testing, and MARL has demonstrated strong capabilities in solving distributed collaboration problems, however, there is still little work reported on deeply integrating these three. Specifically, there is a lack of a systematic framework that can formalise multitasking test scenarios in complex CI/CD pipelines into a standard MARL problem, and utilise environment dynamics extracted from real industrial data for training and validation. Most RL testing studies are still limited to simplified simulation environments or single projects, and the generalisability of their findings is debatable (Hessel et al., 2018). Therefore, this study aims to fill this gap and explore how to build an automated testing framework based on real CI/CD logs and relying on advanced MARL algorithms to solve the problem of collaborative scheduling and decision making of testing tasks in complex software environments, and to promote software engineering automation to a higher level of intelligence and collaboration.

3 Methodology

This section details our proposed framework for automated testing of multi-agent software, as shown in Figure 1. As illustrated in Figure 1, the MASAT framework consists of multiple testing agents (a_1 to a_N), each corresponding to a specific testing task (e.g., unit testing, integration testing). The state extractor generates the global state s_t composed of the code change vector C, resource state vector R, and build history vector H. Each agent observes a partial state of the environment and uses a deep recurrent Q-network (DRQN) to compute its individual action-value function Q_i. The reward calculator computes the global reward based on the current state and joint actions. The QMIX hybrid network combines the individual Q_i values into a joint action-value function, which is used to derive the joint action $\mathbf{u} = (u_1, u_2, \ldots, u_N)$, where u_i denotes the action selected by agent a_i. The system interacts with a simulated Jenkins CI/CD pipeline to train the agents in a high-fidelity environment.

Figure 1 Architecture of the multi-agent system for anomaly testing (MASAT) framework (see online version for colours)



3.1 Problem formalisation: decentralised partially observable Markov decision process-based test modelling

In order to capture the essential characteristics of the CI/CD test environment, we model it using decentralised partially observable Markov decision process (Dec-POMDP). This model is well suited to our problem because it simultaneously considers several key factors: multiple test agents (decentralised decision making), each intelligence can only see a portion of the information about the environment (partially observable, e.g., unit test agents do not directly know the details of the performance tests), the state of the environment changes with the joint actions of the agents (Markovianity), and all the agents share a common end goal (collaborative). We model test task scheduling in the CI/CD pipeline as a Dec-POMDP defined by its tuple $G = \langle N, S, A, O, P, R, \gamma \rangle$, where N: the set of agents, S: the global state space, A: the joint action space, O: the observation space, P: the state transition probability function, R: the reward function, γ: the discount factor.

Intelligentsia and global state: let the set of agents be $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$, with each a_i representing a test task. The global state $s_t \in \mathcal{S}$ is defined as:

$$s_t = (C_t, R_t, H_t) \tag{1}$$

where C_t is the code change feature vector, R_t is the system resource state vector, and H_t is the history build information vector. Specifically, the code change feature vector can include a series of quantitative features such as the number of lines of code modified in this commit, the number of files affected, the entropy of the module distribution of the change, and the type of change analysed based on the commit information (e.g., functionality additions, bug fixes, configuration modifications, etc.).

Observation function: each agent a_i receives a local observation $O_i(s_t)$ at time step t, which is a function of the global state s_t. The observation function O_i is usually a mask or projection that allows the agent to see only the portion of the state relevant to its own task.

State transfer probability: the dynamics of the environment are described by the state transfer function, which defines the probability distribution of transitioning to the next state given the current state and the joint actions of all agents. This probability is approximated by the learned dynamic model in the simulation environment.

Global reward function: the global reward $R(s_t, a_t)$ that the team receives after taking the joint action a_t is designed as:

$$R(s_t, a_t) = R_d + R_c - \lambda_t t_{step} - \lambda_c c_{step} \tag{2}$$

where $R_d = \sum_{d \in D} I_d \cdot w_d$ is the defect discovery reward (I_d is the indicator function and w_d is the defect severity), $R_c = w_{line}\Delta_{line} + w_{branch}\Delta_{branch}$ is the coverage reward, t_{step} is the time consumption, c_{step} is the computational resource cost, and λ_t, λ_c are weight parameters balancing the relative importance of time consumption and resource cost.

The goal of the agents is to maximise the expected discounted return is to maximise the desired discount return:

$$\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k}\right] \tag{3}$$

where $\gamma \in [0, 1]$ is the discount factor, which measures the current value of future rewards.

3.2 MASAT algorithm design

We use the QMIX algorithm, which centers on decomposing the system's joint action-value function $Q_{tot}(\boldsymbol{\tau}, a)$ into a monotonic combination of each agent's action-value function $Q_i(\tau_i, a_i)$.

Individual action-value function: each intelligent body i computes its Q-value based on its action-observation history τ_i through the DRQN network:

$$Q_i(\tau_i, a_i; \theta_i) \tag{4}$$

where $\tau_i = (o_i^1, a_i^1, \ldots, o_i^t)$ represents the action-observation history sequence of agent i, with o_i^t being the observation of agent i at time step t, θ_i is the network parameter of the intelligence i.

Joint action value function: QMIX integrates individual Q-values through a mixing network *mixer* to form joint Q-values:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{a}; \theta_{mix}) = f_{mix}\left(Q_1(\tau_1, a_1), \ldots, Q_n(\tau_n, a_n); \theta_{mix}(s)\right) \tag{5}$$

where $\boldsymbol{\tau}$ is the sequence of histories of all agents, and ϕ is the parameter of the hybrid network. The mixing network parameters are generated by the hypernetwork based on the global state s to ensure the monotonicity constraint described below.

Monotonicity constraint: in order to ensure the consistency of individual strategies with the global interest, QMIX is enforced:

$$\frac{\partial Q_{total}}{\partial Q_i} \geq 0, \qquad \forall i \in \{1, 2, \ldots, N\} \tag{6}$$

This constraint implies that an increase in the Q value of any one of the agents does not lead to a decrease in the joint Q value.

Loss function: the model learns by minimising the temporal difference error:

$$\mathcal{L}(\theta) = \mathbb{E}_{(\tau, a, r, \tau') \sim D}\left[\left(y - Q_{\text{tot}}(\tau, a; \theta)\right)^2\right] \tag{7}$$

where the target value y is:

$$y = r + \gamma \max_{a'} Q_{tot}(\tau', a'; \theta') \tag{8}$$

where r: immediate reward, a': joint action at the next time step, θ and θ' are parameters of the online and target networks respectively. $\mathcal{D}$ is the experience playback buffer.

Strategy derivation: in the execution phase, each intelligence selects an action through a greedy strategy:

$$a_i = \arg\max_{a_{i'}} Q_i\left(\tau_i, a_{i'}\right) \tag{9}$$

Due to the monotonicity constraint, the result of all agents independently choosing the optimal action is the joint optimal action.

3.3 Jenkins-based data processing and simulation environment construction

Feature normalisation: raw numerical features extracted from Jenkins logs (e.g., execution time, memory usage) need to be normalised:

$$x_{norm} = \frac{x - \mu_x}{\sigma_x} \tag{10}$$

where μ_x and σ_x are the mean and standard deviation of feature x on the training dataset, respectively.

Modelling environmental dynamics: we train a feed-forward neural network to predict the next state and immediate reward based on the current state and joint action. The model parameters are learned by minimising the mean square error between predictions and true values.

4 Experimental validation

4.1 Experimental setup

To ensure the reliability and reproducibility of our experiments, we built our simulation environment based on a real public dataset. The dataset is derived from TravisTorrent, a widely used Travis CI build history database (Pinto et al., 2018). The TravisTorrent database records the rich contextual information of each build, including the code commits that triggered the build, the execution status of each stage of the build process (e.g., compilation, different test suites), the elapsed time, changes in code coverage, and the final build results (success/failure). These multi-dimensional historical data provide the data foundation for us to rebuild a simulation environment that reflects the dynamics of a real CI/CD pipeline. We selected 10 of the largest and most complete open-source projects with the most complete build history, covering a wide range of domains such as web frameworks, database middleware, machine learning libraries, etc., and containing a total of more than 58,000 build records. We divided the data into training, validation, and test sets in an 8:1:1 ratio to ensure that the model is evaluated on unseen project build patterns. From these logs, we extracted key fields including build status, phase elapsed time, test results, code change hashes, etc., and used them to build a high-fidelity simulation environment as described in the methodology section.

In order to fully evaluate the performance of MASAT, we have selected four representative baseline methods for comparison. Rule scheduler: this is a fixed-priority based heuristic that executes in a strict 'unit tests → integration tests → system tests' order without skipping any tasks, representing a typical configuration of a traditional CI/CD pipeline (Dimitroulakos et al., 2009). Single-agent DQN: this approach uses a single DQN to make centralised decisions for all test tasks, whose action space is the Cartesian product of the actions of all the agents, and is a direct application of single-intelligence RL to test scheduling (Mnih et al., 2015). MADDPG: this is a state-of-the-art multi-intelligent actor-critical algorithm, where each intelligent has a separate policy network, and is executed through a centralised network. It has an independent policy network, trained through a centralised network of critics, and is a classic baseline in the MARL field (Lowe et al., 2017). Randomised strategy: this baseline randomly selects one available action for each intelligent body at each time step as a lower bound for performance evaluation.

Our evaluation is organised around two core objectives: test efficiency and test quality. Therefore, we employ the following four key metrics: defect detection rate, which is the ratio of successfully revealed known defects (back casted from failure states in the build history) to all injected defects; total test execution time, which is the sum of the simulation time spent completing all the test tasks for the entire pipeline; line-of-code coverage, which is captured through the coverage report of the simulation; and the average cumulative reward, which is used as a direct measure of the learning strategies 's overall strengths and weaknesses.

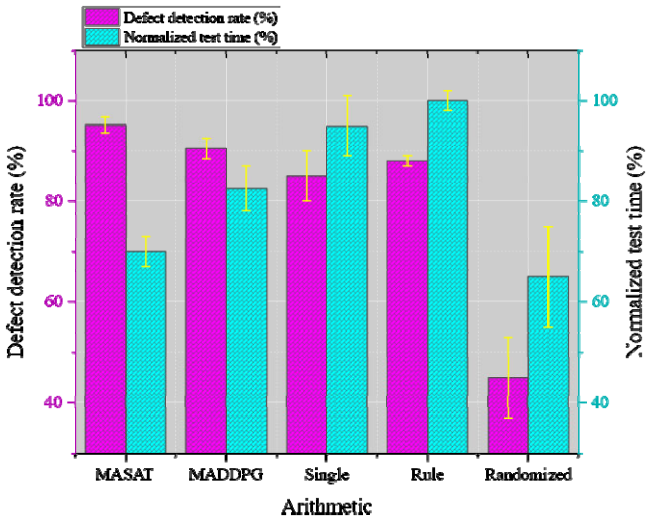
In terms of implementation details, all RL algorithms were implemented using the PyTorch framework. The neural networks in both QMIX and MADDPG use two fully connected layers with a hidden layer dimension of 64. We used the Adam optimiser with an initial learning rate set to 0.001 and linear decay. The empirical playback buffer size is 50,000 and the discount factor γ is set to 0.95. Each experiment is run independently five times and the mean and standard deviation are reported to ensure stability of the results.

4.2 Overall performance comparison

We first evaluated the overall performance of all methods on the test set. Figure 2 shows the comparison results of the methods on two core metrics: defect detection rate and total test execution time. As shown, our proposed MASAT framework (QMIX) achieves the optimal overall performance. In terms of defect detection rate, MASAT significantly outperforms all baseline methods, achieving an average detection rate of 95.2%. A one-way analysis of variance (ANOVA) showed a statistically significant difference in performance between methods ($F(4, 20) = 58.37$, $p < 0.001$). The post hoc Tukey honestly significant difference (HSD) test further showed that MASAT had a significantly higher defect detection rate

than the rule scheduler ($p < 0.001$), the single-agent body DQN ($p < 0.01$), and the randomised strategy ($p < 0.001$), and that the difference with MADDPG reached a significant level ($p < 0.05$). This result validates the advantages of the multi-agent collaboration strategy in pinpointing complex defects.

Figure 2 Performance comparison of different algorithms on defect detection rate and normalised test time (see online version for colours)

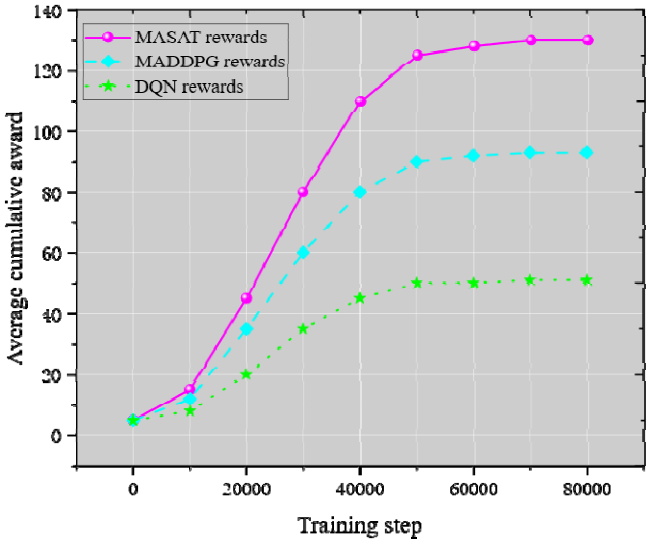


In terms of test efficiency, MASAT also demonstrates its superiority by reducing the total test execution time to about 70% of the baseline rule scheduler. It is worth noting that MASAT does not simply pursue speed, but achieves efficiency gains while ensuring a high defect detection rate. In contrast, although the randomised strategy has the shortest time, its defect detection rate is so low that it has no practical value. The single-intelligence DQN, on the other hand, has an unstable learning strategy due to its large action space and difficulty in capturing inter-task collaboration, resulting in high time consumption and fluctuating defect detection rates. The performance of MADDPG is in between that of the single-intelligence DQN and the QMIX, which may be attributed to the fact that its critic network is not as stable as the monotonic value decomposition mechanism of the QMIX in coping with the complex, higher-order collaborations among the agents in this scenario. To quantify the practical significance of these differences, we computed Cohen's d effect size and found that the effect size of MASAT on defect detection rate relative to the strongest baseline MADDPG, $d = 0.89$, is a 'large' effect, which fully demonstrates that its improvement is not only statistically significant, but also of great practical importance. Cohen's d is a measure of the degree of standardisation of the difference between the means of the two datasets. In general, $d = 0.2$ is considered a small effect, $d = 0.5$ a medium effect, and $d = 0.8$ or more a large effect. The result of $d = 0.89$ in this study indicates that the improvement in defect detection rate of MASAT relative to MADDPG is substantial and highly practicable, implying that the adoption of MASAT is expected to lead to significant quality improvement in a real development environment.

4.3 Convergence analysis

Figure 3 shows the average cumulative reward variation curves of MASAT (QMIX) with MADDPG and single-agent body DQN during the training process. It can be clearly observed that our proposed MASAT framework has faster convergence and higher final performance. In the early stage of training, all methods obtain lower rewards through exploration. After about 10,000 steps, the reward of MASAT starts to rise rapidly and steadily, and basically converges to a high plateau after 50,000 steps. The convergence curve of MADDPG, on the other hand, has significant fluctuations, indicating that its training process is less stable than that of QMIX. The fact that the single-agent DQN has the slowest convergence rate and eventually converges to a much lower reward value than the two multi-intelligent methods is a visual confirmation of the inherent difficulties of single-agent models in dealing with such distributed decision problems. The stable convergence properties of MASAT are crucial for its deployment in real industrial scenarios, as it implies that the algorithm is capable of reliably learning effective collaborative strategies.

Figure 3 Average cumulative reward convergence curves for different algorithms during training (see online version for colours)



4.4 Ablation experiment

In order to investigate the contribution of each component in the MASAT framework, we designed ablation experiments. We constructed two variants of MASAT:

1 MASAT-w/o-Global: in this variant, we removed the input of global state information to the hybrid network, i.e., the parameters of the hybrid network are no longer state-dependent

2 MASAT-w/o-Collab: in this variant, we remove the multi-agent collaboration mechanism so that each intelligence learns independently [i.e., independent Q-learning (IQL) mode] and optimises only its own local reward.

The experimental results are summarised in Table 1. Compared with the full version of MASAT, MASAT-w/o-Global shows a significant decrease in all the metrics, which proves the importance of dynamically adapting the value fusion approach based on the global state, which enables the agents to adopt the most appropriate collaboration mode according to the specific system context. MASAT-w/o-Collab, on the other hand, shows the most significant performance degradation, and its defect detection rate is even lower than that of a single-intelligence DQN, which strongly suggests that the collaboration mechanism at the core of our framework is a key source of performance improvement, and that independent learning that lacks collaboration is unable to cope with the complexity of the dependencies between test tasks.

Table 1 MASAT framework ablation experiment analysis

Methodologies	*Defect detection rate (%)*	*Total test time (minutes)*	*Code coverage (%)*
MASAT	95.2	70.0	88.5
MASAT-w/o-Global	88.1	85.5	82.0
MASAT-w/o-Collab	75.3	92.0	75.8

4.5 Case studies

We selected a specific case for in-depth analysis, which involved a complex build that included a database schema change. The rule scheduler executed all tests in strict order, and when the integration test reported an error due to a failed database connection, it still mechanically executed the subsequent time-consuming end-to-end UI tests, which ultimately resulted in a long total time and failure to pinpoint the root cause of the problem. The MASAT framework, on the other hand, demonstrated superior intelligent decision making: in the unit testing phase, the intelligence responsible for database mapping detected an anomaly and its decision influenced the global policy; subsequently, the integration test intelligence was assigned a higher priority to deeply investigate the database connectivity issue, while the system automatically skipped the UI test, which was highly relevant but not the most critical at the moment. In the end, the MASAT strategy quickly pinpointed the problem to the database connection pooling configuration, reducing the total time spent by 45% over the rule scheduler and avoiding unnecessary resource consumption. This case vividly demonstrates the ability to achieve resource optimisation and rapid convergence of test focus through collaborative sensing and decision making among MASAT agents.

4.6 Experimental results and analysis

The experimental results of this study show that our proposed MASAT framework achieves significant improvement in defect detection rate and testing efficiency compared to existing mainstream methods in automated testing tasks in complex CI/CD environments. These results are not accidental, and there are important theoretical insights and practical significance behind them. First, the success of MASAT fundamentally confirms the rationality and superiority of modelling CI/CD testing as a multi-agent collaborative problem. The core limitation of traditional single-agent-body models (Mnih et al., 2015) or approaches that treat the testing process as a static sequence (Lowe et al., 2017) is the inability to effectively characterise and utilise the complex interactions among testing tasks. MASAT, on the other hand, enables each testing intelligence to not only focus on its own responsibilities, but also perceive and collaborate on global goals through the value function decomposition mechanism (Rashid et al., 2020) through decentralised decision making and centralised learning. This provides a reusable theoretical-technical bridge, namely a value decomposition-based multi-agent collaboration framework, for solving the widespread problem of 'local optimisation and global goal conflict' in software engineering. Our work materialises the principles of 'division of labour' and 'systems thinking' in cognitive theory into a computable and optimisable technical model through the MARL algorithm, realising the leap from theoretical concept to engineering practice.

At the practical level, the MASAT framework shows great potential for landing in real industrial scenarios. The behaviour of the agents in the case study dynamically adjusting the test strategy shows that the system is capable of understanding the build context and making judgements that approximate those of a human expert. This capability means that it can be integrated into existing DevOps toolchains (e.g., Jenkins, GitLab CI) as an intelligent decision-making layer that assist developers in orchestrating test processes. This not only shortens the feedback cycle and speeds up the integration process, but also reduces the cost of fixing defects by pinpointing them more accurately. Teachers and trainers can also utilise these systems to demonstrate modern, intelligent software engineering best practices to their students, thus demonstrating their application and promotion value in education.

However, we must view the limitations of this study with caution, namely the main factors that threaten validity. The primary threat to external validity is that although we used the real-world TravisTorrent dataset, it is after all limited in terms of the types and sizes of projects it covers, and MASAT's performance on larger, more heterogeneous, enterprise-level private projects still require further validation. Future work will evaluate the generalisation ability of the framework through cross-project validation on more diverse datasets, including different programming languages, architectural styles, and on-premises projects, while considering domain-adaptive techniques to enable the

model to quickly adapt to new project environments. Second, is the construct validity, our simulation environment, although built based on real logs, is ultimately an approximation of complex reality. Differences between the simulator and the real CI/CD environment in subtle but potentially critical dynamics (e.g., transient fluctuations in network latency, transient failures of dependent services) may have an impact on the algorithm's final performance. To mitigate the discrepancy between the simulation and real-world CI/CD environments, future work will incorporate real-time data streaming from live CI/CD pipelines (e.g., via Jenkins API) to continuously update the simulation model. Additionally, we plan to integrate domain randomisation techniques during training to enhance the model's robustness to environmental variations. Finally, in terms of the validity of the conclusions, while we have employed statistical tests and effect size analysis to enhance the reliability of the conclusions, the inherent randomness of the RL algorithm requires a larger number of replicated experiments in the future to further solidify the conclusions. To address the inherent randomness of reinforcement learning algorithms, we will conduct larger-scale repeated experiments (e.g., 30+ independent runs) and employ statistical methods such as bootstrapping or Bayesian analysis to provide more robust confidence intervals for performance metrics.

Looking ahead, this research opens up several promising directions whose exploration will lead to substantial improvements in software engineering practices. First, applying more advanced MARL algorithms (such as those explicitly modelling communication) to testing scenarios will enable testing agents to collaborate on complex dependencies through direct information exchange. In practice, this will enable the automated test system to 'consult' like a human test team, and when an intelligent body finds interface anomalies, it can actively notify the test intelligence of the relevant module to carry out targeted validation, thus significantly improving the ability to detect deep defects in complex systems such as microservice architectures, and reducing the problem troubleshooting time. Second, explore the framework's ability to scale in more specialised non-functional testing domains such as security testing and chaos engineering. Specifically, in chaos engineering scenarios, the MASAT framework can be used to coordinate multiple 'fault injection intelligences' that intelligently choose to introduce perturbations in different parts of the system and in different ways (e.g., simulating network latency, service downtime) and observe the system response so as to autonomously learn the most effective fault injection strategy to identify the vulnerable links in the system more efficiently and enhance the overall resilience of the system. Finally, we plan to collaborate with industrial partners to deploy and evaluate the MASAT framework in real enterprise development environments for a long time. The ultimate vision of this research is to transform MASAT from a research prototype to a reliable engineering solution, helping organisations to substantially improve software delivery efficiency and quality. The ultimate vision of this research is to promote software engineering automation from the traditional paradigm of preset rules to a new paradigm of self-adaptation, self-organisation, and group intelligence, and these specific research directions will provide practical and feasible technical paths to realise this vision, contributing to the construction of a smarter and more efficient software development infrastructure in the future.

5 Conclusions

In this paper, we systematically explore the application of multi-agent RL in automated testing of complex software environments and propose an innovative framework called MASAT. The main contributions of this research can be summarised in the following three aspects: first, at the theoretical level, we first rigorously formalise the problem of test task collaboration in the CI/CD pipeline into a Dec-POMDP model, and successfully introduce the QMIX algorithm into this domain, constructing a computable bridge connecting the theory of collaborative cognition and the practice of distributed software testing. Second, at the methodology level, we design and implement a complete set of data processing and simulation environment construction processes based on real Jenkins log data, which provides valuable high-fidelity experimental benchmarks for subsequent related research. Third, at the experimental and practical levels, through rigorous comparison and ablation experiments, we not only verified that MASAT significantly outperforms multiple baseline methods in improving defect detection rate and testing efficiency, but also revealed the intrinsic validity of its intelligent collaborative mechanism through detailed case studies and statistical analyses, fully demonstrating its practical application value and promotion prospects in modern software development processes.

Declarations

All authors declare that they have no conflicts of interest.

References

Arcuri, A. and Fraser, G. (2013) 'Parameter tuning or default values? An empirical investigation in search-based software engineering', *Empirical Software Engineering*, Vol. 18, No. 3, pp.594–623.

Barr, E.T., Harman, M., McMinn, P., Shahbaz, M. and Yoo, S. (2014) 'The oracle problem in software testing: a survey', *IEEE Transactions on Software Engineering*, Vol. 41, No. 5, pp.507–525.

Busoniu, L., Babuska, R. and De Schutter, B. (2008) 'A comprehensive survey of multiagent reinforcement learning', *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 38, No. 2, pp.156–172.

Dimitroulakos, G., Georgiopoulos, S., Galanis, M.D. and Goutis, C.E. (2009) 'Resource aware mapping on coarse grained reconfigurable arrays', *Microprocessors and Microsystems*, Vol. 33, No. 2, pp.91–105.

Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R. and Safina, L. (2017) 'Microservices: yesterday, today, and tomorrow', *Present and Ulterior Software Engineering*, Vol. 9, pp.195–216.

Du, W. and Ding, S. (2021) 'A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications', *Artificial Intelligence Review*, Vol. 54, No. 5, pp.3215–3238.

Faustino, J., Adriano, D., Amaro, R., Pereira, R. and da Silva, M.M. (2022) 'DevOps benefits: a systematic literature review', *Software: Practice and Experience*, Vol. 52, No. 9, pp.1905–1926.

Foerster, J., Assael, I.A., De Freitas, N. and Whiteson, S. (2016) 'Learning to communicate with deep multi-agent reinforcement learning', *Advances in Neural Information Processing Systems*, Vol. 29, p.834.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. and Silver, D. (2018) 'Rainbow: combining improvements in deep reinforcement learning', *Artificial Intelligence*, Vol. 32, p.1.

Humble, J. and Farley, D. (2010) 'Continuous delivery: reliable software releases through build, test, and deployment automation', *Pearson Education*, Vol. 1, p.1.

Lowe, R., Wu, Y.I., Tamar, A., Harb, J., Abbeel, O.P. and Mordatch, I. (2017) 'Multi-agent actor-critic for mixed cooperative-competitive environments', *Advances in Neural Information Processing Systems*, Vol. 30, p.530.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K. and Ostrovski, G. (2015) 'Human-level control through deep reinforcement learning', *Nature*, Vol. 518, No. 7540, pp.529–533.

Moghadam, M.H., Saadatmand, M., Borg, M., Bohlin, M. and Lisper, B. (2022) 'An autonomous performance testing framework using self-adaptive fuzzy reinforcement learning', *Software Quality Journal*, Vol. 30, No. 1, pp.127–159.

Mukherjee, R. and Patnaik, K.S. (2021) 'A survey on different approaches for software test case prioritization', *Journal of King Saud University-Computer and Information Sciences*, Vol. 33, No. 9, pp.1041–1054.

Panichella, A., Kifetew, F.M. and Tonella, P. (2015) 'Reformulating branch coverage as a many-objective optimization problem', *Software Testing, Verification and Validation*, Vol. 2, pp.1–10.

Pinto, G., Castor, F., Bonifacio, R. and Rebouças, M. (2018) 'Work practices and challenges in continuous integration: a survey with Travis CI users', *Software: Practice and Experience*, Vol. 48, No. 12, pp.2223–2236.

Rashid, T., Samvelyan, M., De Witt, C.S., Farquhar, G., Foerster, J. and Whiteson, S. (2020) 'Monotonic value function factorisation for deep multi-agent reinforcement learning', *Journal of Machine Learning Research*, Vol. 21, No. 178, pp.1–51.

Shahin, M., Babar, M.A. and Zhu, L. (2017) 'Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices', *IEEE Access*, Vol. 5, pp.3909–3943.

Stone, P. and Veloso, M. (2000) 'Multiagent systems: a survey from a machine learning perspective', *Autonomous Robots*, Vol. 8, No. 3, pp.345–383.

Sutton, R.S. and Barto, A.G. (1998) *Reinforcement Learning: An Introduction*, MIT Press, Cambridge.

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2015) 'Quality and productivity outcomes relating to continuous integration in GitHub', *Foundations of Software Engineering*, Vol. 9, pp.805–816.

Yoo, S. and Harman, M. (2012) 'Regression testing minimization, selection and prioritization: a survey', *Software Testing, Verification and Reliability*, Vol. 22, No. 2, pp.67–120.

Zhang, K., Yang, Z. and Başar, T. (2021) 'Multi-agent reinforcement learning: a selective overview of theories and algorithms', *Handbook of Reinforcement Learning and Control*, Vol. 9, pp.321–384.