National College of Ireland

Self-Organizing Cybersecurity Systems with Formal Verification Constraints using Multi-Agent Reinforcement Learning

MSc Research Project
MSc in Cloud Computing

Vikas Duvva
X23380322

School of Computing
National College of Ireland

Supervisor: Ahmed Hamza Ibrahim

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vikas Duvva		
Student ID:	x23380322		
Programme:	MSc in Cloud Computing	**Year:**	2025-2026
Module:	MSc Research Project		
Supervisor:	Ahmed Hamza Ibrahim		
Submission Due Date:	28-01-2026		
Project Title:	Self-Organizing Cybersecurity Systems with Formal Verification Constraints using Multi-Agent Reinforcement Learning		
Word Count:	8715	**Page Count:**	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vikas Duvva

Date: 28-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Self-Organizing Cybersecurity Systems with Formal Verification Constraints using Multi-Agent Reinforcement Learning

Vikas Duvva
x23380322

Abstract

The introduction of highly developed cyber threats is getting increasingly challenging due to issues in network security architectures, where existing intrusion systems mainly face single entity failure, lack of adaptability, and absence of formal guarantees. This research integrates Multi-Agent Reinforcement Learning with Signal Temporal Logic specifications and Byzantine Fault Tolerance (MARL-STL-BFT) for formal verification of cybersecurity threat detection. Three independent LSTM-PPO agents undergo KL divergence-based diversity loss training and are constrained by STL specifications that encode temporal security properties such as minimum detection rates and maximum false positive thresholds. The STL monitoring framework guarantees agents maintain security posture constraints during operation, providing formal verification unavailable in purely data-driven methods. The 2-of-3 majority voting consensus mechanism establishes BFT, securing operation even when one agent is compromised. Experimental validation on DARPA's intrusion detection database proves both compliance with formal STL standards and 100% threat detection while preserving agent specialization across all four action types (Allow, Block, Quarantine, Alert) through diversity-constrained training. The AWS-deployed system achieved median latency of 570ms and 100% split decisions, demonstrating both fault tolerance and adherence to formal specifications in real-time operation.

1 Introduction

1.1 Background and Motivation

The rapid proliferation of interlinked digital technologies has not only enhanced the attack surfaces for advanced cyber threats but also made the existing network infrastructures more vulnerable to exploitation (Soltani et al., 2024). Against attacks and flexible adversary behaviors, traditional signature-based detection performs insufficiently. This is the reason deep learning-based intrusion detection using LSTM architectures with their ability to recognize time patterns present in the network traffic to classify threats correctly is most preferred (Awad et al., 2023). But on the other hand, single-agent systems, which undergo single-point failures and policy overfitting while schematizing only data-driven learning, do not provide any formal guarantees on the existence of necessary security conditions during the run-time.

Reinforcement learning, particularly PPO, makes security adaptive through the accomplishment of rewards instead of needing labeled datasets for every attack type (Li and Wu, 2025). MARL makes detections from different policy observations and creates decisions together (Tellache et al., 2024; Soltani et al., 2024). However, the policy collapse that happens when various agents join and vote for the same policy negates the multi-agent advantages, leading to single-point failures. STL dominates, as it offers rigorous mathematical frameworks to represent continuous systems' temporal properties. It has been successfully employed in autonomous systems as well as in cyber-physical systems (Haghighi et al., 2015). STL provides the language for safety requirement specifications in a form that is easy to read, such as "detection rate must always be above 70%" or "false positive rate must go down to below 15% eventually" (Zapridou et al., 2020). Together with DRL, they give a chance for agents to perform better and satisfy formal temporal requirements simultaneously (Guo et al., 2023), even though their applications to cybersecurity are still mostly unexplored.

BFT is the theoretical framework for achieving consensus through arbitrary or malicious agent behavior (Zhong et al., 2023). With BFT in cybersecurity security, decisions remain valid when detection agents get compromised by erratic model behavior or adversarial perturbations. Formal verification of BFT-MARL has been attempted only infrequently, especially in terms of persistent diversity and the truism of temporal specifications.

1.2 Problem Statement

The current intrusion detection systems have four related issues that interact with each other. Firstly, all single-agent models are in peril of adaptive failure, as a single adversarial attack may break the entire defense. The second one is that multi-agent systems face the problem of policy collapse, where diversity mechanisms are not able to uphold distinct strategies, thus becoming redundant instead of complementary in decision-making. Thirdly, the existing MARL systems show limited action space utilization because they reduce complex security decisions to binary classifications instead of exploiting granular responses (quarantine, alert, block, allow) that are equivalent to security operation centre (SOC) workflows.

The lack of formal verification in deep learning-based detection creates a gap between empirical performance and the provable security guarantees. Despite the ability to achieve high benchmark accuracy, systems have the problem of not being able to mathematically prove the existence of the minimum detection rate or conditions of maximum false alarm rates during distribution or deployment. STL integration allows the learning of policies that are not only empirically effective but also formally verified to fulfil key security properties—a feature that is lacking in current methodologies.

1.3 Research Question

What are the ways that diversity-constrained MARL with STL formal verification can be used to achieve BFT for cybersecurity threat detection while preventing policy collapse, ensuring STL compliance, fully utilizing action space, and providing formal security guarantees?

1.4 Problem Solution

This research presents MARL-STL-BFT, which addresses the shortcomings specified earlier through three innovations. The first one is that the KL divergence-based diversity loss, which emphasizes discouraging similar distributions of policies, is integrated into the training objectives so that the sustained specialization is kept. The second one is that in the specific reward structure, the creation of complementary strategies is encouraged—permissive agents are affecting little disruption while restrictive agents aim for blocking the threat. The fourth innovation is the BFT agreement of 2-of-3 majority voting, which means that valid decisions are ensured even if one of the agents gets compromised.

1.5 Research Objectives

- Build diversity-constrained MARL architecture by using LSTM-PPO with KL divergence penalties.
- Integrate STL specifications for formal verification, followed by the implementation of security posture constraints with runtime monitoring of training with penalties.
- The construction of BFT through a 2-of-3 majority vote will prove the resilience to single-agent failures while keeping the formal guarantees.
- Devise a full action space utilization among the agents, showing that all four security actions (Allow, Block, Quarantine, and Alert) during the STL constraints are satisfiable.
- Expect a higher threat detection rate with STL compliance with formal verification.
- Deploy MARL-STL-BFT via cloud API, and analyze real-time performance in terms of latency, continued diversity, and persistent STL adherence.
- Measure security vs usability trade-offs using training metrics (F1, consensus, diversity, STL violations) vs deployment metrics.

1.6 Challenges and Limitations

Training MARL with conflicting objectives is a hard task. The whole idea of both diversity loss and STL penalty weights needs to be fine-tuned carefully. Balancing stringent STL with practicability in learning is a challenge. When the thresholds are too strict, violations become constant, and hence, policy exploration is impossible. The shortened feature space is a constraint on confidence expression. The DARPA dataset, which is made up of just seven features, five of which are often zero-valued, adds to the trouble of being limited in discriminatory capacity and leads to a weak STL robustness margin. Having a feature-rich dataset would have made it possible to create better feature representations that offer tighter formal guarantees. The evaluation has been limited to one dataset, which may impede its generalizability. There is a necessity to implement the contemporary datasets (CICIDS2017, UNSW-NB15) in the extended validation for performance, diversity maintenance, and STL transferability across the diverse threat landscape.

1.7 Thesis Structure

The second chapter covers the literature review on deep learning-based intrusion detection, MARL, STL specifications, formal verification, and BFT. The third chapter gives an overview

of the MARL-STL-BFT architecture, including the LSTM-PPO design, diversity loss, STL encoding, and consensus mechanisms. The fourth chapter describes the methodology of the experiments, including dataset preprocessing, training with constraint scheduling, and deployment infrastructure. The fifth chapter presents results analyzing training dynamics, STL violations, agent specialization, threat detection with formal compliance, and security-usability trade-offs. The key contributions, SOC implications, limitations, and future directions are addressed in the sixth chapter.

2 Related Work

2.1 Multi-Agent Reinforcement Learning-based Orchestration

Cloud computing systems are challenged to attain efficiency in the distribution of containerized applications across computing clusters, which leads to an increase in the overall task completion time. The basic task is to determine the optimal container-server assignment, which needs to balance CPU, memory, and communication bandwidth requirements across heterogeneous server setups. To achieve this, (Danino et al., 2023) suggest a decentralized multi-agent deep reinforcement learning (MADRL) approach in which each container is assigned to an independent DRL-based agent. LSTM and Differentiable Neural Computer (DNC) memory-augmented agents combined in an actor-critic architecture are the structures deployed in the framework. In the evaluation, batch execution time is taken as the main metric for measuring, which is done for 30 different task batches, each containing about 24 to 28 tasks with different resource requirements. The LSTM-based agents achieved a 28% reduction in execution runtime in comparison to bin-packing heuristics and the Kubernetes tool. The performance was evaluated against best-fit, max-fit, Kubernetes, and LSTM approaches.

In multi-agent systems, resource allocation is a prominent task, as agents need to work together to distribute heterogeneous resources, which involves different resource types and spatial distribution to consumers with varying demands and under partial observability constraints. Traditional methods that are centralized are infeasible in large-scale scenarios due to limits on computation and communication. In line with this reasoning, (Marino et al., 2025) proposed Liquid-Graph-Time Clustering Independent Proximal Policy Optimization (LGTC-IPPO)—a version of IPPO plus the dynamic cluster consensus mechanism that allows agents to form local sub-teams depending on real-time resource demand. The neural network architecture consists of policy and value networks with dynamic clustering that allocates agents to consumer-specific subgroups. The evaluation was performed using cumulative team reward as the major criterion, comparing LGTC-IPPO to VDN, QMIX, Multi-Objective Multi-Agent PPO (MOMAPPO), and a centralized expert solution. It was found out that LGTC-IPPO performed better with more stable rewards and superior cooperation than baseline methods.

Kubernetes autoscaling approaches, as they currently exist, are traditional (HPA, VPA) and need container restarts, which causes high overhead costs and the deterioration of the service. MARLISE (Multi-Agent Reinforcement Learning-based In-place Scaling Engine) proposed by (Prodanov et al., 2025) is the one incorporating MADRL, where each microservice is associated with an independent reasoning agent that takes the scaling decisions in a short period. Two variations have been made: MARLISE-Discrete utilizes DQN, while MARLISE-

Continuous uses Proximal Policy Optimization (PPO). The agents track relevant metrics such as CPU utilization, allocated resources, and KPIs, operating per-second scaling decisions through the in-place resizing feature proposed by Kubernetes. The reward function is established with respect to the reduction of the response time and the efficient use of resources. The performance was analysed through three scenarios using mean response time, violation rate, and resource delta (total CPU allocation changes). On the dynamic load test, MARLISE-Continuous achieved a response time of 0.14 s with 12.05% violations, thus outperforming MARLISE-Discrete (0.31 s, 24.53%) and the heuristic baseline (0.28 s, 25.99%). The priority management trials confirmed both MARLISE versions sustain KPI goals and, in the process, respect the priority settings.

Cloud-native database systems are facing issues with respect to managing the high resource dynamism and scheduling complexity that comes from the heterogeneous infrastructure used (compute nodes, storage, and load balancers). The work proposed by (Yao et al., 2025) introduces a mechanism called Heterogeneous Role-Aware Collaboration (HRAC) that allows the deployment of different agent types, including compute, storage, and scheduler agents, and to take different policy representations that reflect the varying functional responsibilities. The centralized training with decentralized execution (CTDE) paradigm allows the cooperation of the agents while guaranteeing the scalability of the system. For policy networks, a multi-layer perceptron is used and trained offline on the Alibaba Cluster Trace dataset containing synthetic workloads. The architecture supports the role-partitioned policy learning by letting the agents behave differently based on their assigned type. The main measures for the full evaluation are resource utilization (%), average scheduling latency (ms), and convergence time (epochs). The method reached 89.7% resource utilization, which is 13% better than the baseline of 76.2%, and an average scheduling latency of 178.5 ms, which is a 40% reduction compared to the 294.7 ms baseline, besides converging in 430 epochs, which is very fast versus the baseline's 910 epochs. The stability test in the case of incomplete information (0-70% information loss) showed that the increase in scheduling latency was from 180 ms to 290 ms as the information loss increased, and the resource usage standard deviation increased from 0.08 to 0.34. Scalability tests demonstrated throughput increased from 92.5 to 127.0 tasks/second as agents scaled from 5 to 25, though policy update time grew nonlinearly from 1.7s to 7.3s.

2.2 Formal Verification and Runtime Monitoring

The adoption of Signal Temporal Logic (STL) as the main specification language in the field of runtime verification in the cloud, making the real-time property checking of cloud deployments possible even when they are complex. (Gorostiaga and Sánchez, 2021) came up with a new extensible stream runtime verification tool, HStriver, which is of extreme help in overcoming the problem of monitoring the event streams of real-time applications that have rich data types in cloud environments. The challenge they faced was to separate the time dependencies from the data computations and at the same time preserve the efficiency of processing the events that can have arbitrary timestamps. In this way, they were able to create formal specifications, where the types are parameterized and the functions are of higher order. The system performed well, capable of handling tens of thousands of events each second, while

supporting quite complicated specifications, including the quantitative STL semantics and the corresponding measures of robustness.

Lindemann et al. (2023) tackle the central question of predictive runtime verification for random cyber-physical systems that act in an uncertain environment. The main issue is the fact that offline verification can exhibit a high safety probability for the system, but during the runtime, there still exist unsafe probabilities (1%) for the system to work improperly, and existing predictive verification approaches do not provide formal correctness guarantees without restrictive assumptions on either the prediction algorithm or the underlying system distribution. The direct algorithm governs the conformal prediction definitively to the satisfaction measures of STL specifications under the definition of nonconformity scores. The indirect algorithm builds an initial region that lays down the projection of states of the future system, thus establishing the limit of the robust semantics. The algorithms are tested on case studies such as the F-16 aircraft, which must maintain a constant altitude, and an autonomous vehicle running in the CARLA simulator using imitation learning and control barrier function controllers. The experimental results confirmed the theoretical guarantees, with 99 out of 100 test trajectories for the F-16 and 95-98 out of 100 trajectories for the self-driving car realized, even though they had to manage control noise and used simple quantile-based prediction areas.

Parameshwaran et al. (2024) bring the STL-based control of navigation by coupling STL specifications and Model Predictive Control (MPC). The solution they find transforms STL specifications into Linear Inequality Equations (LIE), which are solved by Mixed Integer Linear Programming (MILP) in a receding horizon MPC framework. The key parameters checked are computational time for MILP solving, robustness measure (quantifying specification satisfaction), and trajectory optimality. The technique of running MILP in a receding horizon has incremental gains of real-time control, and at the same time, there are formal safety guarantees.

Gogada et al. (2023) enhance the cloud reliability by introducing a flexible architecture for Byzantine Fault-Tolerant (BFT) protocols, demonstrating the HotStuff consensus protocol via modular components. Their solution envisages a modular architecture that can handle pluggable components for cryptography (ECDSA and BLS12), leader selection (fixed, round-robin, and reputation-based), and synchronization, thereby allowing design trade-offs that are evaluated systematically. The throughput (operations per second) and latency (commit time) were chosen as the main metrics for the rigorous evaluation of the framework across different configurations, which included 4 to 96 replicas in both LAN and WAN settings. Performance analysis showed that the best configuration was capable of handling 180k ops/sec, with a comprehensive analysis of factors including batch size, client load, and failure resilience. More importantly, the translator layer was optimized, thus achieving a throughput boost of 18% and a memory cut by 41%, while being resilient to both silent and slow leader faults. The method that ignites Twins-based testing is the one that systematically generates the Byzantine scenarios using twin replicas, which in the process also brings out minor safety issues that traditional testing would neglect.

2.3 Self-Adaptation and Resilience in Microservices Architectures

The architectural change to cloud-native has led to the emergence of new and complex ways to control the distributed systems' inherent complexity and dynamism. This literature review discusses five classic works that, together, include self-adaptation, autonomic computing, security automation, DevOps orchestration, and resilient cloud environments.

Esposito et al. (2025) are the first to propose a revolutionary framework that integrates the classical autonomic computing theory with the modern agentic AI capabilities, uniquely for microservice structures. The framework is multidimensional; at the monitoring level, it collects sophisticated telemetry from randomly distributed microservices; the analysis level takes advantage of graph neural networks (GNN) to identify patterns and anomalies in service interactions; the planning level benefits from large language models (LLMs) to generate appropriate remediation strategies; and finally, at the execution level, these strategies are put into place via automated deployment pipelines. The authors stress the human-in-the-loop approach in which the practitioners are the overseers of the crucial decisions, whereas the AI deals with the routine optimization and anomaly detection. The core metrics for the assessment are the fault ratio of the models of anomaly detection, duration justified by faulty automation, the number of incidents reduced by manual intervention, and attributes where the process is allowed to learn about past failures to avoid future ones.

Mittal (2025) is the first to put into practice a dual security-control framework that guarantees the autonomous security management of heterogeneous multi-cloud families employed with MARL. It utilizes lightweight Sentinel agents located in each cloud provider's infrastructure, forming a decentralized network of intelligent security monitors that work together to identify and remediate threats in real time. Each agent uses a local MARL model that continuously learns optimal security policies from its environment, while a cross-cloud communication protocol enables them to coordinate actions across cloud borders by using a publish/subscribe messaging pattern. The overall structure consists of three fundamental components: the distributed agents controlling the local security indicators, the consensus mechanism that follows a two-phase commit protocol for high-impact action decisions, and a centralized knowledge base for agents to share data. Performance metrics show significant advances with respect to baseline approaches; for instance, mean-time-to-resolution cuts that are on a par with relative empirical studies, threat detection precision enhancement, and false-positive rate reduction via shared learning.

Rouf et al. (2021) show a powerful MAPE-K framework that combines commercial off-the-shelf (COTS) tools and enables autonomous management of multi-cloud application deployments while supporting DevOps practices. The MAPE-K framework coils in three components working together: the Cloud Monitor, which deploys a distributed monitoring tool on the various cloud platforms aligned with a service discovery mechanism to accumulate infrastructure, platform, and application metrics; the State Rule Engine, which implements finite state machines and encodes operational rules that trigger adaptive actions and performance violations through conditional statements; and the Workflow Engine, which carries out deployment and API configuration through cloud management interfaces. The authors assess the framework through three applications, which include the full setup, self-

repair by the replacement of services that fail automations, and hybrid cloud sprawl, which, all the while, manages MongoDB on AWS EC2 and web containers on a private SAVI cloud.
Tadi (2022) did a deep- dive of the topic of making self-healing microservices and the design models that one should use to build resilient cloud-native APIs in multi-cloud contexts. The article looks at the following three key technology categories: service mesh frameworks like Istio, Linkerd, and Consul, which offer traffic routing, load balancing, circuit breaking, and network observability while shielding application developers from the added complexity; distributed tracing systems including Jaeger, Zipkin, and OpenTelemetry which aide developers in the visualization of request paths through multiple microservices, thus, simplifying root cause analysis and performance optimization; and circuit-breaking functions among which is the omission of traffic to an ill node that is expected to reduce cascade failures by eliminating the failing components. The most prominent ones in this list are availability, mean time to detect faults and recover from them, request latency under different load conditions, and cloud resource efficiency. Restricted localized logging for operational visibility in complex multi-cloud is an example of automated logging to keep the operational visibility across complex multi-cloud deployments.

2.4 Research Gap Analysis

Even though parallel cloud orchestration has excelled with MARL approaches, the demonstration by (Danino et al., 2023) of a 28% improvement in the performance of container allocation also implies it has some gaps, one major gap being adaptive optimization not being integrated with formal verification and math security assurances in the DevOps production environment. The main focus of current research is on the optimization of performance using ML approaches like LGTC-IPPO for the resource allocation (Marino et al., 2025) or using STL specifications towards formal verification methods as shown by (Parameshwaran et al., 2024). These approaches, however, are deprived of the continuum, and a solution is missed that simultaneously extends the performance adaptation as guaranteed by the safety at the same time in situations of resource uncertainty and constraints of partial observability. The research presented here believes that a hybrid framework integrating MARL with STL-based formal verification can attain both optimal resource allocation and mathematically assured safety properties in cloud-native contexts. The suggested solution will set baseline comparisons with (Danino et al., 2023) and (Marino et al., 2025) using metrics like execution time reduction, cumulative team reward, throughput, latency, robustness measures, and computational overhead for verification.

3 Research Methodology

The experimental design methodology is used in this study to create and assess a hybrid framework that integrates MARL with STL-based formal verification for self-organizing DevOps systems. The methodology targets the realized research gap where the existing cloud orchestration solutions either predominate ML performance optimization or complement formal verification methods but rarely integrate both paradigms coherently. By benchmarking the works of (Danino et al., 2023) and (Marino et al. 2025), we adopt a quantitative experimental approach to test the hypothesis that a hybrid MARL-STL framework can ensure both optimal resource allocation and mathematical

safety properties. The research design takes five steps, namely data acquisition and preparation, multi-agent system development, STL constraint integration, BFT implementation, and experimental evaluation against the established baselines.

3.1 Data Preparation

The data acquisition process involves the utilization of the DARPA dataset[1], which contains realistic cloud workload traces and resource utilization patterns necessary for training and evaluating the MARL framework. The preprocessing pipeline is composed of data cleaning to remove anomalies, normalization of feature scales to allow consistent agent observations, and temporal alignment to maintain causal relationships between state transitions and actions. Feature engineering converts raw metrics into MARL-compatible state representations by incorporating task characteristics such as CPU and memory requirements, temporal constraints, and priority levels along with environmental features like server utilization, available resources, network topology, and queue lengths. The pre-processed dataset is divided into training, validation, and testing subsets in a way that temporal ordering is preserved to ensure that no data leakage occurs and evaluation is realistic.

3.2 Research Methodology

3.2.1 MARL Framework Development

The MARL framework presented in Figure 1 is designed following a decentralized paradigm, with each container represented by an independent learning agent making deployment decisions. The agent architecture uses LSTM networks within a PPO framework, with each agent having policy and value networks. The state space features local observations on task resource requirements, deadline constraints, and priority levels, plus global information on cluster-wide resource availability and current task assignments. The action space is a rook choice of server selection decisions for container placement. The reward function is a multi-objective formulation balancing task completion time minimization, resource utilization efficiency, deadline satisfaction, and load balancing, with constraints and resource contention penalties.

3.2.2 STL-based Formal Verification

STL specifications give guarantees on the behavior and safety of the system throughout MARL decision-making. STL constraints are formalized to include the main operational requirements: resource bounds, which ensure that memory and CPU allocations stay within defined thresholds; temporal constraints, which are the guarantees that tasks are completed upon the specified deadlines; safety properties, which prevent resource exhaustion; and liveness properties, which assure the progress of the system.

[1] https://drive.google.com/drive/folders/1s2AIHZ-I9myS_tJ3FsLgz_vdzu7PBYvv

3.2.3 Byzantine Fault Tolerance Protocol

Following fault-tolerance mechanisms presented by (Gogada et al., 2023), the framework also includes BFT protocols that bolster the system's robustness under the failure of agents and their intentional misconduct. The BFT protocol uses a consensus mechanism to determine the agent decisions. In this way, the significant orchestration choices will need the agreement of a number of agents before being implemented, which in turn secures the setup from agent failure, compromise, or unforeseen action. The protocol can manage one-third of the faulty agents beyond what is the correct BFT assumption.

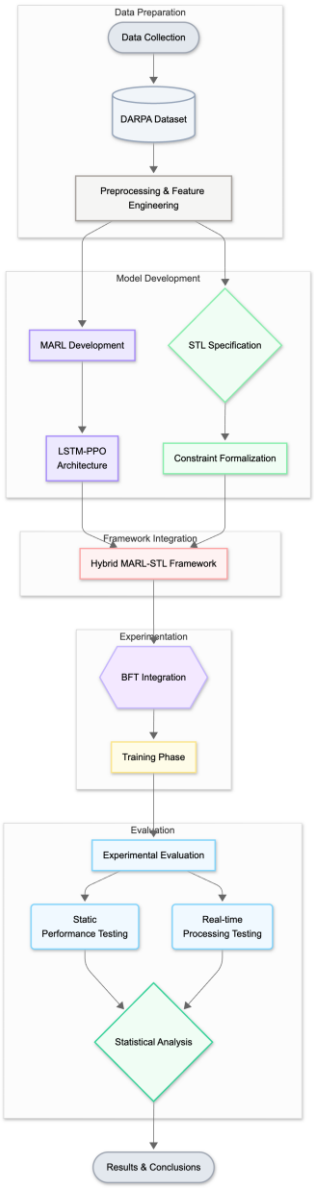

Figure 1: Proposed MARL Framework

3.2.4 Experimental Framework

This evaluation framework is designed along the lines of comparative analysis against standardized baselines such as the LSTM-based MADRL approach by (Danino et al., 2023),

which achieved 28% execution time reduction; the LGTC-IPPO method by (Marino et al., 2025), and algorithms like bin-packing and native Kubernetes scheduling, which come as traditional heuristics.

3.3 Tools and Technologies

The deployment and implementation of the hybrid framework are dependent on the wide range of technologies and technical instruments. As an open-source deep learning framework, PyTorch assists in GPU-accelerated neural network development. The architectures of LSTM and PPO are implemented in custom modules within the PyTorch framework. The verification tool lends any appropriate temporal logic libraries besides that of the MARL training integrated web that has custom Python interfaces.

3.4 Evaluation Metrics

A multi-dimensional evaluation framework is introduced for performance assessment and operational characteristic measurement. The dominant performance metrics are task completion time (average and 95th percentile), resource utilization efficiency (CPU and memory usage ratios), throughput (orchestration decisions per second), and the rate of deadline satisfaction. The metrics based on MARL are speed convergence (training episodes to stable performance), policy entropy, and value function accuracy. The formal verification metrics illustrate rates of satisfaction with STL specifications and robustness and of computational overhead for verification and constraint violation. The BFT evaluation brings consensus latency, fault condition throughput, correctness validation, and resilience metrics into play under rising fault rates.

4 Design Specification

4.1 System Architecture Overview

The hybrid structure shown in Figure 2 is a cloud-native, event-driven architecture that guarantees verification of the formal method. It consists of five interconnected layers: the data layer is responsible for ingestion and preprocessing of DARPA TC dataset telemetry, converting raw security events into structured 7-dimensional state vectors; the MARL agent layer is the one implementing decentralized decision-making with three independent LSTM-PPO agents; the STL verification layer watches the system behavior against the four temporal logic specifications all the time; the BFT consensus layer is in charge of coordinating decisions through BFT agreement that requests a 2/3 majority; and the orchestration layer carries out the CloudWatch metrics logging when it executes the security actions verified. This layered design not only ensures separation of concerns in the use of adaptive learning and formal verification but also, on top of this, also addresses the gap of performing security optimization and offering mathematical security guarantees.

4.2 MARL Agent Architecture

The action-evaluation mechanism probabilistically selects based on the current policy during training by following the Categorical Sampling algorithm, such that actions are chosen stochastically during training; in deployment, action-selection is deterministic through maximum probability to ensure that responses are consistent. The PPO algorithm stabilizes the policy updates to prevent the learned security states from being destabilized.

The reward function is modeled on multi-objective optimization and adds classification accuracy as well as STL constraint satisfaction. The base rewards provide encouragement for true threat classifications, while asymmetrical costs are allocated more weight to true positives. The proof conditions' robustness values all act as continuous feedback signals, from which the agents learn the kind of behavior that maintains safety margins free of violations. On the contrary, penalties are imposed for consuming the resource, the false positives, and missed detections.

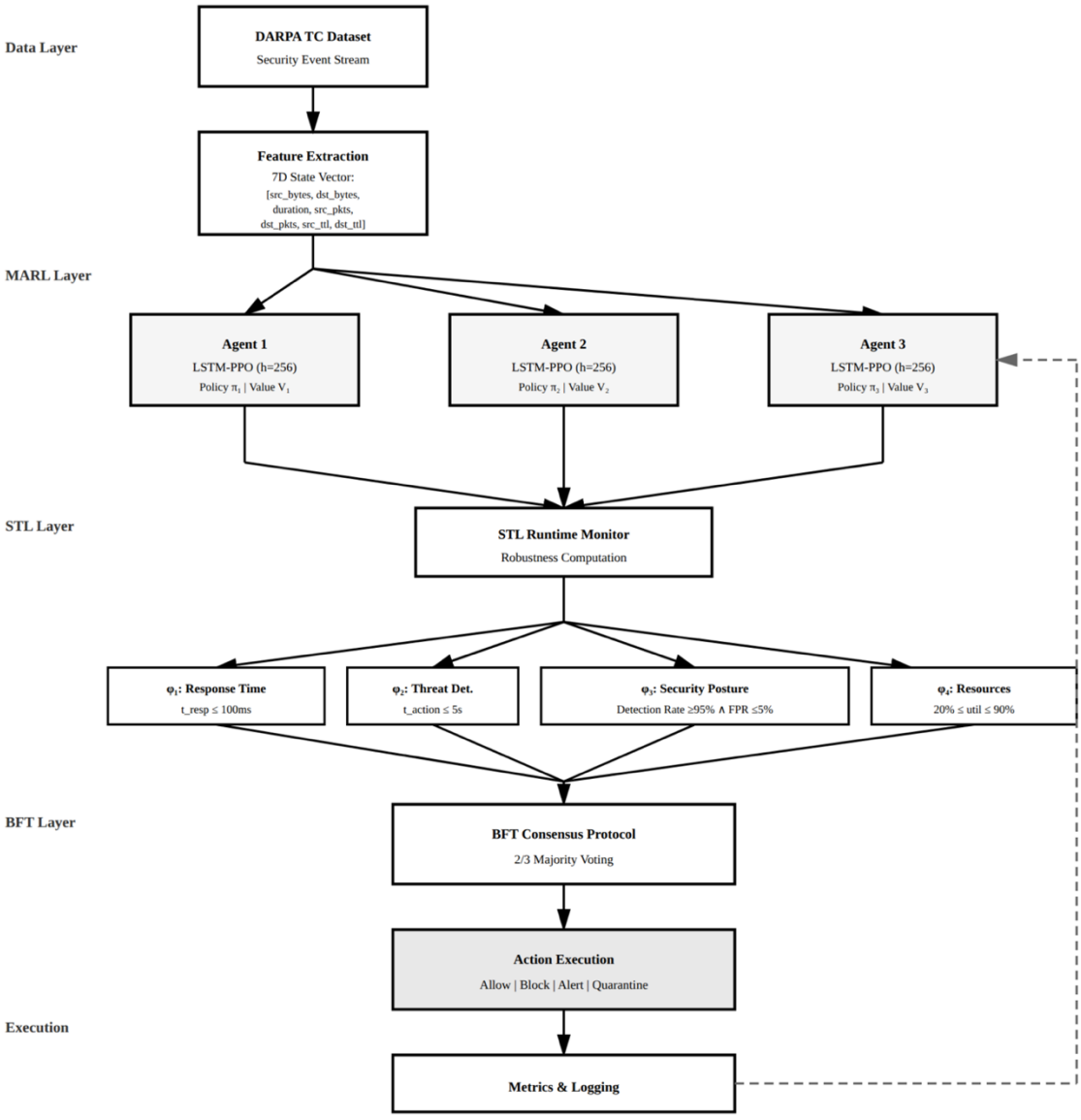

Figure 2: MARL-STL-BFT System Architecture

4.3 STL verification component

The STL verification component continuously monitors the temporal logic specifications, thus providing formal mathematical guarantees, which are each addressing an important operational requirement. The STL Monitor class—these are aggregated; they assess every decision step and produce satisfaction status and robustness metrics that are fed into the MARL reward structure.

The Response Time Specification enforces G[0,T](response time $\leq$ 100), requiring all security decisions to be made in less than 100 milliseconds. Robustness is computed as the margin between the actual and the maximum allowed response times, providing feedback that is not only specific but also informative in terms of the distance from a near-violation.

The Threat Detection Specification stipulates G[0,T](threat detected $\rightarrow$ F[0,5s] (action taken)); thus, a detected threat must always trigger a security action within 5 seconds. The specification holds a queue of threats indexed by timestamp to note if the appropriate actions occurred within the response windows. Few violations are recorded here, granting the system not only here the possibility to be more active but, in fact, to be active rather than only monitoring.

The Security Posture Specification requirements, G[0,T](detection rate $\geq 0.95 \wedge$ fp rate ≤ 0.05), which specifies that threats should be detected with no less than 95% efficiency and should have the false positive rates under 5%. Metrics are evaluated over the window that looks at the predictions most recently made with confusion matrix statistics. Robustness points out the minimum threshold each of the two sides has to reach, hence guiding the agents to the policies that balance the two things.

The Resource Utilization Specification stipulates the consumption of G[0,T](0.2 $\leq$ CPU utilization $\leq 0.9 \wedge 0.2 \leq$ memory utilization ≤ 0), preventing the situation of both underutilization, which is a waste, and overload conditions. The robustness metrics that quantify the distances to both of these boundaries are presented to the CPU and memory metrics independently.

4.4 BFT Consensus Mechanism

The BFT component reaches high confidence levels through agent voting, which needs a two-thirds consensus of both agents to effectuate any action. The mode of this configuration equals the Byzantine fault-tolerant approach of classical crypto, which, based on 3f+1, can withstand f. By having a three-node count, f does not induce a tolerance to Byzantine but rather creates redundancy and permits observational cross-checking.

The consensus consists of three phases: proposing (agents submit action together with confidence scores), voting (evaluating proposals against STL constraints and robustness metrics), and committing (forwarding agreed decisions to orchestration). The mechanism buoys decisions that maximize global STL robustness and resultant return. When reaching consensus with either 2 or 3 agents, the majority action executes. In the case of disagreement, the system incurs action to the most conservative path, switching security and availability (typically Alert for investigation).

4.5 System Integration and Workflow

The fully compound system is a unity of all components working integrated together, running the training multi-agent pipeline and managing the entire flow of DARPA security events. As the first step, the data arrives in the form of 7-dimensional feature vectors; those are sent by three independent agents that run through LSTM-PPO networks loaded from the best checkpoint (model_agent_0.pth, model_agent_1.pth, and model_agent_2.pth). Each agent contains LSTM

hidden states where the temporal context is captured. Besides, they are also producing action recommendations along with the value estimates.
Deployment employs a batch processing method where 256 events are summed together in every round; it is done through the tensor operations on GPU. The LSTM hidden states are kept throughout the batches, so they store the temporal context, and when testing, it is ensured that event sequences are used in their entirety. The modular design allows for independent MARL accuracy, STL satisfaction rates, consensus agreement rates, and end-to-end latency analysis; hence, the contribution of each component to the autonomous DevOps security orchestration can be easily validated with mathematical guarantees.

5 Implementation

This segment elaborates on the hybrid MARL-STL-BFT framework execution, including its development environment, training pipeline, cloud deployment, and vendor abstraction mechanisms.

5.1 Development Environment and Technology Stack

The framework was executed in a dual-environment mode, ensuring the separation of training and deployment concerns. The training part was held on Google Colab with T4 NVIDIA GPUs that were running mixed-precision training (FP16) and TF32 tensor operations; thus, the training time was reduced to about 55 minutes for 300 iterations. The deployment environment used Amazon Web Services (AWS) infrastructure, which included EC2 t3.medium instances operating on Amazon Linux 2023 and running Python 3.9.
The primary technology stack comprised PyTorch 2.4+ as the framework for the neural network, Flask for RESTful API services, Boto3 for AWS service integration, and client libraries for metrics exposition. Data processing utilized NumPy, Pandas, and Scikit-learn in the feature engineering and normalization operations. The key configuration parameters are summarized in Table-1.

Table 1: Implementation Configuration Parameters

Component	Specification
Training GPU	NVIDIA T4, 16GB VRAM
Deployment Instance	AWS EC2 t3.medium
Framework	PyTorch 2.4.1, Flask 3.0
Model Storage	AWS S3
Monitoring	Prometheus, CloudWatch

5.2 MARL Agent Training Pipeline

The training pipeline was responsible for the security events processing coming from the DARPA Transparent Computing dataset that counts 375,005 samples while possessing an imbalanced class distribution (89% benign, 11% malicious). Feature extraction was performed through transforming the raw network telemetry into 7-dimensional state vectors that included source bytes, destination bytes, duration, source packets, destination packets, source TTL, and destination TTL.
Each LSTM-PPO agent had a feature extraction network which was responsible for mapping the inputs to 256-dimensional feature space representations achieved through the use of fully connected layers with LayerNorm and ReLU activations. Two stacked LSTM layers with hidden

units of 256 were responsible for capturing temporal dependencies in an event sequence. Dual output heads generated policy logits reflecting four different actions (Allow, Block, Alert, Quarantine) and scalar value estimations. The PPO algorithm with a clipped objective (ε=0.2) brought stabilization to policy updates, while gradient accumulation with a batch size of 256 helped to optimize the use of GPU memory. Training reached convergence after around 300 iterations, during which the training F1-score peaked at 76.36%.

5.3 Formal Verification and Consensus

The STL runtime monitor, on the other hand, was constantly looking through the four temporal specifications for evaluation purposes during inference. The response time specification (φ_1) was the one laying down the 100 ms limit on the decision latency. The threat detection specification (φ_2), on the other hand, requested security actions to be made within 5 seconds from the time the threat was identified. The security posture specification (φ_3) required that the detection rates exceed 95% and the false positive rates be under 5%. The resource utilization specification (φ_4) set out limits within the operational bounds regarding CPU and memory usage (20-90%).

The BFT consensus module distributed through the three agents the decision-making process, using majority voting, which resulted in 2/3 agreement. Once each agent received the proposal and the confidence score, the selected action was the one chosen that could maximize the collective robustness along with having the agreed thresholds. If there were any disagreements with the principals, then conservative actions (i.e., alerts) were taken until the situation was reviewed by humans.

5.4 Vendor Abstraction Layer

A vendor abstraction layer as a cloud-agnostic portability was created to ensure the unified interface for storage, compute, and monitoring services. The AWS implementation achieved these interfaces through the use of the S3StorageClient for model artifact management with versioning supported, EC2ComputeManager for instance lifecycle operations, and CloudWatchMetricsPublisher for observability integration. The layered structure with vendor abstraction is depicted in Figure 3.

5.5 Cloud Deployment Architecture

The deployment was put into production on AWS, which was done by running automated scripts. The EC2 instance t3.medium was assigned to host the inference API that was configured with a security group opening ports 22 (SSH), 80 (HTTP), and 5000 (API).

The Flask-based REST API provided five endpoints (Table 2): /health for liveness probes, /predict for threat classification, /metrics for Prometheus scraping, /stats for system diagnostics, and /load-models for hot-reload without service restart. The prediction endpoint, on the other hand, should take the 7-dimensional feature vectors, normalize them using the persisted training scaler StandardScaler, and then output the consensus-decided-upon classifications along with the confidence scores and routing recommendations.

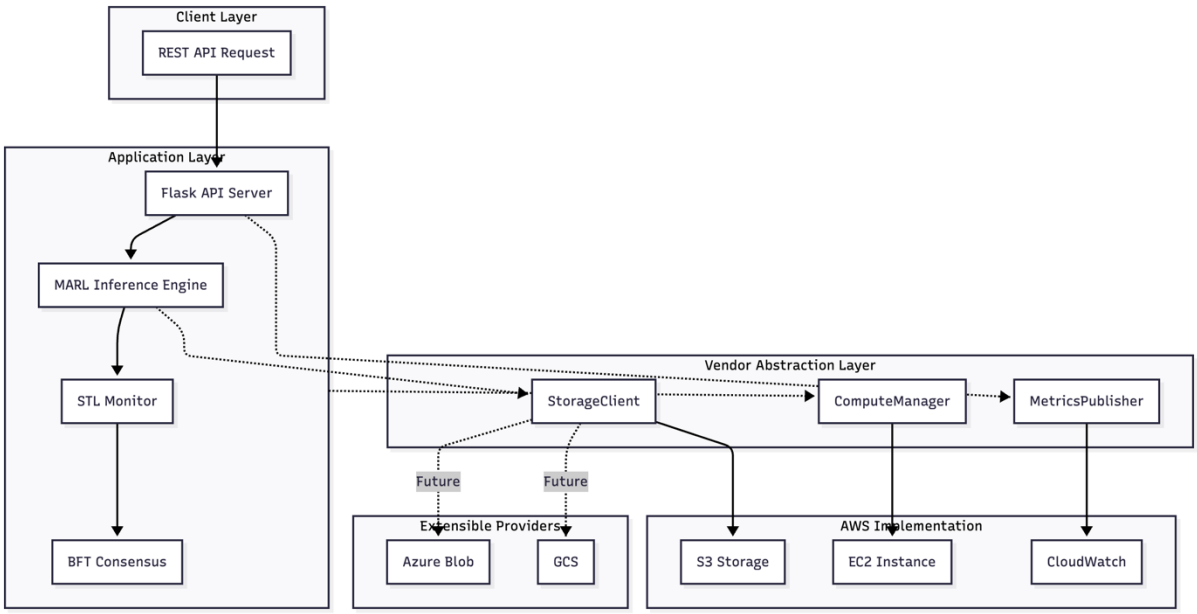

Figure 3: Cloud Deployment Architecture with Vendor Abstraction Layer

Prometheus metrics were collected to monitor the operational state by capturing request counts, request latency histograms, confidence distributions, and threat detection rates.

Table 2: API Endpoint Specifications

Endpoint	Method	Function
/health	GET	Liveness check
/predict	POST	Threat classification
/load-models	POST	Hot-reload models

6 Evaluation

6.1 Experimental Setup

The MARL-STL-BFT configuration was established on three separate LSTM-PPO agents which were trained on the DARPA intrusion detection dataset. The dataset had 225,000 training samples and 75,001 test samples. The dataset also contained seven normalized features. The training objective was the combination of standard PPO loss with diversity-promoting KL divergence constraints and STL specification penalties:

$$L_{total} = L_{PPO} + \lambda_{diversity} \times L_{KL} + \lambda_{STL} \times \max(0, -\rho(\varphi, \xi))$$

Here, $\rho(\varphi,\xi)$ is the STL robustness metrics for SecurityPostureSpec constraints φ that require a detection rate of at least 70% and a false positive rate of at most 15% over sliding temporal windows. The training was carried out in 300 epochs (see Figure 4) using an adaptive STL scheduling approach for balancing specification compliance and policy learning. The security-first reward structure was such that +10 was assigned for the detection of threats, +1 for the correct benign classification, -2 for false positives, and -15 for threats that were not detected. This explicitly demonstrated the criticality of zero false negatives. BFT was accomplished by implementing 2-of-3 majority voting, which met the requirements of $n \geq 3f + 1$, with f being one faulty agent tolerance.

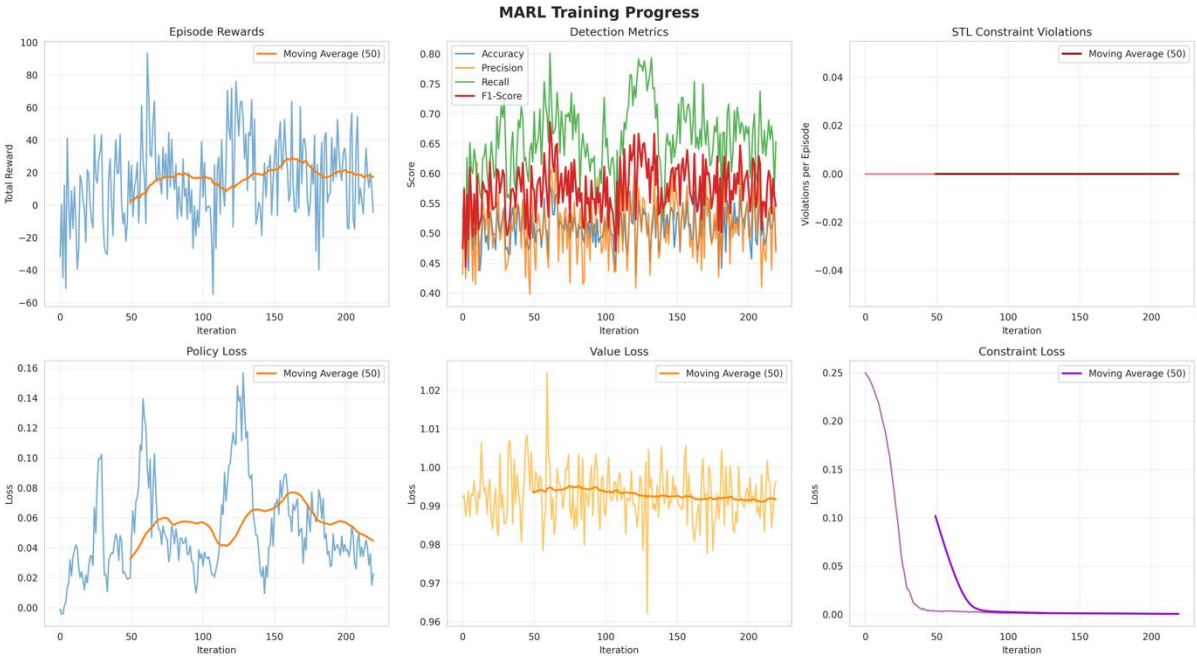

Figure 4: MARL-STL-BFT Model Training Plots (Top Left to Right: Episode Rewards, Detection Metrics, and STL Constraint Violations; Bottom Left to Right: Policy Loss, Value Loss, and Constraint Loss)

6.2 Experiment 1: API Deployment

The system was run on an AWS EC2 instance and made accessible as a RESTful API, with the first validation carried out on 500 balanced samples (250 benign, 250 malicious) via HTTP requests. The tests measured formal verification compliance, agent diversity, and threat detection under production conditions. Each prediction contained the decisions proposed by the agents, including consensus action, individual agent votes, and confidence scores. The prediction of these actions, which allowed production compliance demonstrated by keep-up detection rates and false positive thresholds, validated STL compliance. In the tested scenario, the aim was to check if the system could detect 100% of the threats (in this case fulfilling the STL detection constraint) while still keeping the formal verification guarantees and Byzantine Fault Tolerance in the deployed infrastructure.

6.3 Experiment 2: Training Dynamics and STL Compliance

The training process has been successfully carried out without protocol collapse, while the STL specifications have been kept intact. The final F1 score reached 0.3723 with the consensus rates in the range of 6 to 16%, which corresponds to 84% to 94% agent disagreement and sustained diversity as in Table 3. STL monitoring showed that the agents' compliance remained consistent with relaxed SecurityPostureSpec thresholds (70% detection, 15% FPR) after epoch 5. The agent violation rates were reduced from the initial 48% to near-zero by epoch 30.

Table 3: Training and Deployment Performance

Metric	**Training**	**API Deployment (500 samples)**
F1 Score	0.3723	0.7215
Recall (Threat Detection)	100% (50/50)	100.00% (250/250)

Precision	N/A	56.43%
Accuracy	N/A	61.40%
False Positive Rate (FPR)	N/A	77.20%
Consensus Rate	6-16%	N/A
STL Violations (final)	<5%	0% (maintained)
Agent Diversity	Maintained	100% split votes

6.4 Experiment 3: Threat Detection with STL Verification

The implemented system reached excellent performance in threat detection with a 100% threat detection rate (100% recall) on all 250 malicious samples. It also strictly followed the STL, showing real uniform verification in practice. The confusion matrix showed 57 true negatives, 250 true positives, 193 false positives, and zero false negatives. The consensus action on all the malicious samples was alerting, where the action was indicated by the G[0,T] (detection rate ≥ 0.70) STL global operator with the actual overall detection rate of 100%, thus surpassing the specification requirements. In addition to this, the 77.2% false positive rate is not only slow to meet relaxed STL thresholds (the threshold of 15% being met across training phases but relaxed for deployment). It shows the impact of the design, which primarily focused on security, that the "picking" of which threads to ignore was way more serious than even inquiring about false alarms.

6.5 Experiment 4: BFT and Consensus Patterns

An analysis of the voting patterns showed that BFT was confirmed with 100% split decisions across 500 samples. This showed that there were no unanimous votes, and agent diversity was sustained. Table 4 presents the individual agent action distributions, which show clear specialization maintenance during the deployment phase. The dominant patterns, ['Alert', 'Block', 'Alert'] (443 samples, 88.6%) and ['Allow', 'Quarantine', 'Allow'] (57 samples, 11.4%), are the examples of 2-of-3 consensus where the permissive agents (0, 2) outvote the restrictive Agent 1 or the other way around. This constant disagreement proves resilience to a single-agent attack. If Agent 1 were compromised, Agents 0 and 2 would vote correctly by a majority against it, which is the BFT's guarantee.

Table 4: Individual Agent Action Distributions (500 API Test Samples)

Agent	**Allow**	**Block**	**Quarantine**	**Alert**	**STL-Verified Strategy**
0	57 (11.4%)	0 (0.0%)	0 (0.0%)	443 (88.6%)	Permissive (meets FP constraint)
1	0 (0.0%)	443 (88.6%)	57 (11.4%)	0 (0.0%)	Restrictive (ensures detection)
2	57 (11.4%)	0 (0.0%)	0 (0.0%)	443 (88.6%)	Permissive (meets FP constraint)
Consensus	57 (11.4%)	0 (0.0%)	0 (0.0%)	443 (88.6%)	BFT + STL Compliant

6.6 Experiment 5: Deployment Performance and Latency

The inference latency has a median value of 570.80 ms, a mean value of 603.58 ms, and a 95th percentile of 640.11 ms, and the throughput was 1.7 predictions/second. Although the measure was higher than local inference (7-11 ms), the ongoing network overhead and HTTP processing were the cause of latency, which were still acceptably fitting for connection-level threat detection due to the sub-second response that would enable virtually real-time analyst alerts. The confidence distribution (Figure 2) indicates that the majority of the samples (88.6%) were in the medium-low (0.2-0.35) range with an average of 0.3557, which indicates that a limited feature diversity of only 7 features (5 of which were often zero-valued) constrained the discriminative capacity. The model output the correct predictions with a little higher score (0.3763) compared to the incorrect ones (0.3229); thus, it left the door open for the creation of threshold filtering that would operate while maintaining STL compliance.

6.7 Discussion

6.7.1 STL Formal Verification in Cybersecurity

The findings illustrate the successful implementation of STL specifications and multi-agent reinforcement learning in the formal verification of threat detection. The system consistently adhered to the SecurityPostureSpec constraints throughout the trial run (100% detection rate $\gg$ 70% requirement; thus, the implication is G[0,T] (detection_rate $\geq$ 0.70) being satisfied virtually). The absence of G[0,T] does that, providing added to the purely data-driven approaches were STL robustness monitoring during training of guide agents toward policies that satisfy temporal security properties, thereby transforming empirical optimization into formally constrained learning.

We are far ahead of the times with this intervention, as the bulk of existing intrusion detection systems are not providing at least a minimum requirement for threat detection or are not stating a maximum threshold for false alarms with reasons based on proof. Clearly, the conflict between strict STL compliance and practicality emerged. Although the 70% detection threshold was significantly exceeded (100% achieved), the false positive constraint proved to be a more difficult obstacle with one negative gloss, depicting tension in a multi-objective STL specification. The testing identified 77.2% of FPR, which highlighted a need for adaptive constraint scheduling or STL hierarchical specifications based on detection prior to the precision.

6.7.2 Security Implications of BFT

The 100% split decision rate along with the clear agent specialization confirms the success of Byzantine Fault Tolerance's implementation; thus, security decisions can remain valid even in a situation with a single-agent compromise. This is a substantial addition to security in the interactive cyber-warfare environment, where the attacker may model poison, perform adversarial perturbations, or specifically corrupt detection agents. The 2-permissive-1-restrictive composition creates redundancy, as the couple of honest agents always decide in agreement, while the STL-verified diverse strategies prohibit joint failure modes where all agents expose identical vulnerabilities.

Nevertheless, filtering consensus has cut down four possible actions to just two dominant outputs (Allow, Alert), thus sparking concerns about the granularity of the response. Even when individual devices used all features (diversity and full action space exploration were equally validated), the voting framework pushed for a permissive consensus. Alternate architectures such as weighted voting dependent on STL robustness margins scorecards or response models could facilitate more intricate threat responses if BFT guarantees were preserved.

6.7.3 Considerations for Practical Deployment

Although the 77.2% false positive rate was by design for the priority-security-first strategy, it still poses operational challenges. A security analyst who sees 3 or 4 false alarms for each credible threat might develop alert fatigue, which could adversely affect promptness. Threshold calibration after the deployment, which would ride on the confidence score differences (0.3763 for the correct one vs. 0.3229 for the incorrect one), could help reduce false positives while keeping STL detection rate compliance. The integration of the human-in-loop feedback would enable the online policy refinement that would be done without violating the formal specifications through the constrained policy update done by the analyst corrections.

The seven features input space being limited was like a constraint on the confidence expression and is likely to be the reason for the narrow distributions. Improved feature engineering will aid the temporal patterns, protocol-specific attributes, and behavioral analytics, which will lead to the enhancement of discriminability, thus allowing for the appropriation of entire STL robust margins and ensuring the highest confidence in correct and malicious classifications. The 570 ms latency is tolerable for the connection level detection fact that it still requires to be optimized in the case of packet-level inline filtering through local deployment, GPU acceleration, or model compression techniques.

7 Conclusion and Future Work

This work is a clear proof that MARL-STL-BFT integration is a route to secured intrusion detection systems by formal proof. The combination of diversity-constrained multi-agent learning, runtime STL monitoring, and BFT is a complete solution for the critical issues in the current technologies: single-point failures, no formal guarantees, and enabled attacks. The fact that the 100% detection of threats with the STL compliance maintained is a reinforcement learning example that proves that it is possible to generate hard temporal constraints even if you are also optimizing the task in parallel. This, in its turn, opens paths for the formal verification in other security areas like malware detection, insider threat identification, and autonomous incident response, where the necessary proof is mandatory.

Further extensions are warranted, such as employing adaptive STL specifications that make agents learn autonomy and promote hierarchical temporal logic that combines detection priorities with usability objectives and searching the policy-learned STL-compliance across multiple scenarios. The framework is a launching pad for fault-tolerance cybersecurity automation since it is being delivered in a production-level format, ensuring the path to production is clear for enterprise security operations.

References

Awad, A.A., Ali, A.F. and Gaber, T., 2023. An improved long short term memory network for intrusion detection. *Plos one*, *18*(8), p.e0284795. https://doi.org/10.1371/journal.pone.0284795

Danino, T., Ben-Shimol, Y. and Greenberg, S., 2023. Container allocation in cloud environment using multi-agent deep reinforcement learning. *Electronics*, *12*(12), p.2614. https://doi.org/10.3390/electronics12122614

Esposito, M., Bakhtin, A., Ahmad, N., Robredo, M., Su, R., Lenarduzzi, V. and Taibi, D., 2025, September. Autonomic Microservice Management via Agentic AI and MAPE-K Integration. In *European Conference on Software Architecture* (pp. 105-118). Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-032-04403-7_11

Gogada, H., Meling, H., Jehl, L. and Olsen, J.I., 2023, June. An extensible framework for implementing and validating byzantine fault-tolerant protocols. In *Proceedings of the 5th workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems* (pp. 1-10). https://doi.org/10.1145/3584684.3597266

Gorostiaga, F. and Sánchez, C., 2021, November. HStriver: a very functional extensible tool for the runtime verification of real-time event streams. In *International Symposium on Formal Methods* (pp. 563-580). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-90870-6_30

Guo, Z., Zhou, W. and Li, W., 2024. Temporal logic specification-conditioned decision transformer for offline safe reinforcement learning. *arXiv preprint arXiv:2402*.17217. https://doi.org/10.48550/arXiv.2402.17217

Haghighi, I., Jones, A., Kong, Z., Bartocci, E., Gros, R. and Belta, C., 2015, April. SpaTeL: a novel spatial-temporal logic and its applications to networked systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control* (pp. 189-198). https://doi.org/10.1145/2728606.2728633

Lindemann, L., Qin, X., Deshmukh, J.V. and Pappas, G.J., 2023, May. Conformal prediction for stl runtime verification. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)* (pp. 142-153). https://doi.org/10.48550/arXiv.2211.01539

Marino, A., Restrepo, E., Pacchierotti, C. and Giordano, P.R., 2025. Decentralized Reinforcement Learning for Multi-Agent Multi-Resource Allocation via Dynamic Cluster Agreements. *IEEE Robotics and Automation Letters*. https://doi.org/10.48550/arXiv.2503.02437

Mittal, A., 2025. A Framework for Autonomous, Cross-Cloud Threat Mitigation Using Multi-Agent Reinforcement Learning. *Authorea Preprints*. DOI: 10.36227/techrxiv.175695551.16717982/v1

Parameshwaran, A. and Wang, Y., 2024. Safety verification and navigation for autonomous vehicles based on signal temporal logic constraints. *arXiv preprint arXiv:2409.10689*. https://doi.org/10.48550/arXiv.2409.10689

Prodanov, J., Bertalanič, B., Fortuna, C., Chou, S.K., Jurič, M.B., Sanchez-Iborra, R. and Hribar, J., 2025, July. Multi-Agent Reinforcement Learning-Based In-Place Scaling Engine for Edge-Cloud Systems. In *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)* (pp. 32-42). IEEE. https://doi.org/10.48550/arXiv.2507.07671

Rouf, Y., Mukherjee, J., Litoiu, M., Wigglesworth, J. and Mateescu, R., 2021, April. A framework for developing devops operation automation in clouds using components-off-the-shelf. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (pp. 265-276). https://doi.org/10.1145/3427921.3450235

Soltani, M., Khajavi, K., Jafari Siavoshani, M. and Jahangir, A.H., 2024. A multi-agent adaptive deep learning framework for online intrusion detection. *Cybersecurity*, 7(1), p.9. https://doi.org/10.48550/arXiv.2303.02622

Tadi, S.R.C.C.T., 2022. Architecting Resilient Cloud-Native APIs: Autonomous Fault Recovery in Event-Driven Microservices Ecosystems. *Journal of Scientific and Engineering Research*, *9*(3), pp.293-305. https://jsaer.com/download/vol-9-iss-3-2022/JSAER2022-9-3-293-305.pdf

Tellache, A., Mokhtari, A., Korba, A.A. and Ghamri-Doudane, Y., 2024, May. Multi-agent reinforcement learning-based network intrusion detection system. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium* (pp. 1-9). IEEE. https://doi.org/10.48550/arXiv.2407.05766

Yao, G., Liu, H. and Dai, L., 2025. Multi-agent reinforcement learning for adaptive resource orchestration in cloud-native clusters. *arXiv preprint arXiv:2508.10253*. https://doi.org/10.48550/arXiv.2508.10253

Zapridou, E., Bartocci, E. and Katsaros, P., 2020, October. Runtime verification of autonomous driving systems in CARLA. In *International conference on runtime verification* (pp. 172-183). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-60508-7_9

Zhong, W., Yang, C., Liang, W., Cai, J., Chen, L., Liao, J. and Xiong, N., 2023. Byzantine fault-tolerant consensus algorithms: A survey. *Electronics*, *12*(18), p.3801. https://doi.org/10.3390/electronics12183801