Minimax Theory and its Applications

Volume 11 (2026), No. 3s, 108 to 122

Robust Minimax Optimization of Security Risks for Real-Time Kubernetes Configuration Management

Dr. Jayashree S. Patil[1*], Bhageshwari Ratkal[2] ,Dantu Venkata Lalitha Parameswari[3], Kommanaboyina Lakshmi Sundara Soujanya[4], Gandham Sandhya[5], Md. Bushra Yaseen[6]

[1,2,3,4,5,6]Computer Science & Engineering, G Narayanamma Institute of Technology & Science, Shaikpet, Hyderabad, India

Email: jshivshetty@gnits.ac.in[1], bhagya.ratkal@gnits.ac.in[2], dvlalitha@gnits.ac.in[3], drklssoujanya@gnits.ac.in[4] , sandhya@gnits.ac.in[5], byaseen.com@gmail.com. [6]

Abstract

Kubernetes has emerged as the standard software that automates the deployment, scaling and management of containerized applications within cloud-native infrastructures. Nevertheless, its loose and declarative YAML model of configuration carries a high attack surface of malconfigurations, such as the absence of security contexts, unlimited resource assignments and privilege hyperpylifications. These kinds of misconfigurations often lead to security holes, poor performance, and cluster instability. The available solutions like the static analysis tools and large language models (LLM) frameworks like GenKubeSec seek to identify insecure settings through Kubernetes manifest analysis. However, such methods are usually external advisory and do not enforce directly into the Kubernetes control plane, and use computationally costly models, which restricts their applicability in lightweight and latency-sensitive deployments. In order to overcome such shortcomings, the current paper introduces KubeGuardian, a real-time and intelligent Kubernetes protection system to enforce policies and measure risks. The implementation of KubeGuardian as a validating admission controller is done to intercept deployment requests before these are received into the cluster. This framework takes a deterministic approach to policy enforcement with Open Policy Agent (OPA) alongside dynamic risk scoring with lightweight and interpretable machine learning models conditioned on empirically observed patterns of misconfiguration. Along with the prevention of unsafe deployments, KubeGuardian offers context-based remediation recommendations and displays compliance insights with an interactive administrative dashboard. The system facilitates the current DevSecOps, such as GitOps-driven processes and live warning systems. KubeGuardian offers scalable, effective, and production-ready security governance over Kubernetes environments, according to the experimental results.

Keywords: Cloud-Native Security, Risk Scoring, Machine Learning, DevSecOps, Admission Controller, Open Policy Agent, Policy-as-Code, Kubernetes Security..

1. Introduction

Kubernetes is the preferred orchestrator system for the deployment, scaling, and operation of containerized applications on cloud-native infrastructure, marking the end of cloud-native infrastructure. Kubernetes makes the lifecycle management of the application in the distributed environment easy due the fact that it decouples the underlying hardware resources and allows them to be protagonisticallydeployed by using a collection of configuration files written as YAML. Its design enables the developers to define the state of the intended system and the platform reconcile the state of actual state of the cluster therefore constituting scalability, fault tolerance and scalability in operation [2].

Although the above are advantages of Kubernetes, it has big attack surface because of its complexity in configuration model and flexibility. There are many security sensitive parameters of Kubernetes manifests such as level of container privilege, resource quota, network access control, and identity settings. Some of the most common causes of Kubernetes security breaches include, though are not limited to, privileged containers, the lack of security contexts, unchecked allocations of resources, unsafe volume mounts, and excessive role-based access control (RBAC) permissions [8]. Operational Kubernetes deployments as studied in the real world indicates that over half of production manifests contain one or more critical security misconfigurations that place their users at the risk of unauthorized access, privilege creep, and compromise of their clusters [11].

However, the suggested tools are some of the currently existing to solve these issues, and they include Checkov, KICS, kube-bench and kube-score among others, which are a few in nature of all the tools used in the security

field of state. Each of these tools may be used to check the Kubernetes manifests and infrastructure-as-code artifacts against already defined security policies and possible violations and noncompliance cases [4]. These can be used in the validation of baseline mechanism, as an offline scanner or as a CI/CD pipeline validation, but they do not impose on the Kubernetes control plane. This leaves them incapable of ensuring that the unsafe settings are not implemented in real-time especially during the fast DevOps implementation [13]. Further, rule-based methods will tend to generate high false-positive rates, and will not have any contextual data of deployment intention.

The new paper has referred to the deployment of Large Language Models (LLMs) to facilitate the security analysis of Kubernetes through semantic reasoning and the contextual awareness. The other systems like GenKubeSec [9] and LLMSecConfig [16] have been quite effective in identifying, searching and correcting errors in configuration of Kubernetes. These systems are rather precise and intensive in recall using domain specific knowledge and retrieval augmented reasoning. The proposed solutions on the foundation of LLM are however, computational in nature and they need large amounts of inference and are usually treated as an outwards motion system and not an enforcement one. This means that they cannot be as much applicable to real-time control of admissions and resource limited cases.

One of the inherent flaws of the current Kubernetes security solutions is the inability to integrate with Kubernetes API lifecycle, as a natural consequence. The majority utilize the tools to operate following the deployment or even as the pre-deployment advisors so as to make sure that the unsafe manufactures are included into the cluster before it is identified. Moreover, the existing solutions are either rule enforcement procedures that are deterministic or inference procedures that are probabilistic that do not integrate hard compliance and adaptive risk awareness. This is the gap that causes the necessity of the common framework that is capable of enacting the needed security policy yet would dynamically evaluate the risk with regard to the situation at deployment.

In order to address these shortcomings, the present paper suggests KubeGuardian, a smart Kubernetes-based security framework, a system, that integrates real-time policy enforcement with the machine learning-based risk scoring system.

The name of a validating admission controller is KubeGuardian that intercepts the deployment request before a deployment request is admitted to the cluster via the Kubernetes API server. The model consists of a deterministic validation of policy using Open Policy Agent (OPA) and small and interpretable machine learning models to detect minor misconfigurations and quantify the risk of deployment [6]. The policy is an added value in the form of KubeGuardian where the enforcement is directly on the control plane and, therefore, the non-compliant or high-risk settings are blocked even before it can affect the security of the cluster.

In addition to the enforcement KubeGuardian offers total observability which has a graphical administrative dashboard. The violations of the policy are displayed on the dashboard, risk score and previous admissions decisions are displayed so that the administrators could monitor the security posture and highlight the tendency of typical misconfigurations to emphasize remediation. KubeGuardian is efficiency and scalability focused and the lightweight machine learning models and micro-services are contrasted with the systems founded on the LLM which are founded on the heavy inference pipelines. The structure is also scalable to development of current DevSecOps procedures such as Gitops pipelines and auto alerting to ensure compliance and administration.

The most notable contributions of this work are:

- Test and Implementation of real time Kubernetes admission control system with both OPA based policy enforcement and machine learning based risk scoring.
- Creation of simplified and understandable risk assessment platform in the course of determining configuration-level security threats.
- The enforcement and risk visualization and visualization are implemented unanimously in the Kubernetes control plane.
- The experimental findings that show positive misconfiguration detection, low admission latency, and enhance governance visibility

KubeGuardian will go a step higher to Kubernetes security to be proactive, scalable, production-ready by enforcing deterministic policy and adaptive intelligence. Section 2 will include the review of the work related; Section 3 will include the description of the system architecture and functional modules, Section 4 will include the description of the workflow and the implementation, Section 5 will include the description of the results of

the experiment and the final part of the paper will include the concluding remarks about the paper itself and the future directions of the research.

2. Related Work

The Security in Kubernetes and containers environment has been on the forefront as the complexity of cloud-native deployments has increased and configuration-based vulnerabilities are prevalent. The current state in the literature on this topic area can be broadly separated into systems of rule-based statical analysis, policy-as-code enforcement, machine learning-based risk assessment, and systems based on large language models (LLM) into misconfiguration detection and remediation systems.

Early efforts to secure Kubernetes deployments primarily relied on the ideas of rule scanning and static analysis. Checkov, KICS and kube-bench are tools that test Kubernetes manifests and infrastructure-as-code artifacts against established security policies in order to identify any mal-configurations related to container privileges, network exposure, and resource utilization [4]. These are quick and deterministic to verify, but they possess disadvantages in the shape of rigid definition of rules and lack of context awareness. It has been established that even under the assumption of the tools used being static, the rate of false positivity may be extremely high, and may fail to identify multi-resource or cross-layer misconfigurations, which are reported only by considering manifests holistically [13]. Moreover, these tools do not typically run on the Kubernetes control plane, and only offer advisory feedback, and not a security guarantee.

To address the governance and compliance problem of a cloud-native architecture, policy-as-code (PaC) tools such as Open Policy Agent (OPA) and Gatekeeper have been suggested [6]. These frameworks allow organizations to define declarative policies using domain-specific languages like Rego and run the policies consistently across deployment pipes and runtime environments. According to Di Martino et al., PaC promotes the auditability, reproducibility, and governance by enacting security controls on the system workflows [6]. However, PaC frameworks are not founded on adaptive intelligence and are only grounded on deterministic policies, which is not sufficient since they are not able to detect emerging and progressive misconfiguration patterns. It becomes increasingly difficult to up keep exhaustive and up-to-date sets of policy as clusters are augmented and work load diversified.

The idea of machine learning application to support security analysis in distributed and configuration-driven systems has been explored in up-to-date research. Bufalino et al. [2] demonstrated that systematic classification of configuration errors with the help of empirical datasets can lead to a significant increase in detection accuracy. Similarly, Racharla et al. [12] also constructed a preprocessing and classification pipeline of the software defects detection and how the feature extraction and pattern learning played a vital role in the defects contexts. Even though the models are not Kubernetes-specific, their frameworks influence the development of risk assessment models founded on ML directly depending on the design of the manifest analysis. However, some of the ML-based solutions are offline classifiers and are not associated with the real-time deployment processes.

With the emergence of the large language models, some studies have already been conducted on the analysis of the Kubernetes security of LLM. Ye et al. [16] proposed a framework (LLMSecConfig) that is based on retrieval-augmented LLM reasoning to detect and repair misconfigurations of containers, but not the application semantics. The success of the framework in its repair was very high since it involved both the symbolic reasoning and contextual inference. Similarly, GenKubeSec proposed by Malul et al. [9] is an end-to-end LLCM-based system that performs misconfiguration detection, localization, explanation, and remediation. The outcome of the experiment was high accuracy and recollection as compared to the conventional stationary tools. Despite these good results, LLM-based system is computationally costly, introduces inference latency and is not typically incorporated as an advisory system. Their use in real-time controls of admissions and resource bound production clusters is limited by the fact that they need bulky models.

Configuration security of Helm charts and reusable Kubernetes templates is the direction of other researches. Chen et al. [4] found that many of the Helm charts they had studied had a high proportion of fixable misconfigurations, most of which are maintained by a fear of doing any harm to application semantics. Even when it is combined with the application of the static analysis, the intelligent contextual reasoning can be applied to decrease the number of false positives in the process of analysing Helm-based deployments, as it was demonstrated by Minna et al. [10]. These findings demonstrate the importance of remediation processes that are both conservative and explainable- which is largely disregarded by automated security tools.

Besides the analysis at the manifest-level, some studies have observed the network-level risks and runtime risks

in Kubernetes and containerized systems. Bufalino et al. [2] proposed a taxonomy on misconfigurations of Kubernetes networks and demonstrated how minor configuration errors may result in subsequent lateral movement and privilege escalation. Tayori et al. [15] developed CORAL which is a logical model that creates attack graphs using the composition of configuration data and real time telemetry to quantify risk.

Even though such techniques are captivating rich contextual knowledge, they are computationally expensive and demand large volumes of data when running thus limiting their application in latency sensitive admission techniques.

Adaptive risk assessment and intelligent decision-making also support hybrid security architecture research. The necessity of balanced and diverse machine learning system decisions not to rely on the prevalent patterns was also highlighted by Balduzzi et al. [1], and it is directly applicable to the Kubernetes misconfiguration-detecting case. Similarly, Ganguli et al. [6] and Jansang et al. [9] have also indicated the problems of scalability, performance benchmarking and dynamic behavior of distributed systems further reinforcing the need of lightweight and readable security mechanisms.

In general, the existing Kubernetes security providers possess several constraints: they implement the deterministic rules of policy-as-code frameworks, they apply the rules off-line when using the ML-based methods, and they incur the high costs of computations and inapplicability of the enforcement of the systems based on the LLM-driven frameworks into practice. In addition, most of the current solutions are external advisory tools rather than integration into an enforcement mechanism of a Kubernetes API lifecycle. The presence of these gaps creates acceleration to the implementation of a single framework that incorporates the real-time implementation of the policy and dynamic assessment of interpretable risks.

3. Proposed Methodology

This section contains the proposed KubeGuardian framework along with the modules of design rationality, system architecture and functionality. The methodology is developed in a manner that it offers real-time security enforcement to Kubernetes deployments by integrating deterministic policy validation and adaptive risk assessment. KubeGuardian, in comparison to the traditional external-only tools that are utilized in a conventional static analysis enables admission control to be performed to prevent insecure settings during deployment.

3.1 Design Objectives and Methodological Overview

The principal aim of KubeGuardian is to provide real-time, smart and enforceable management over Kubernetes settings in terms of security. The primary principles of the methodology will be the following:

- **Real-time enforcement:** Security checks must occur before manifests are admitted into the cluster.
- **Hybrid validation:** Deterministic policy enforcement must be complemented by adaptive risk analysis.
- **Low-latency operation:** The adoptions decisions will comply with the requirements of Kubernetes API server time.
- **Interpretability:** Risk decisions should be explainable to developers and administrators.
- **Scalability:** The structure must have the capacity to deal with huge and dynamic clusters.

In order to reach these goals, KubeGuardian combines Open Policy Agent (OPA)-based statical validation and a lightweight machine learning (ML)-based risk scoring system, orchestrated by a Validating Admission Controller. The hybrid approach is consistent with the previous studies that propose integrated deterministic and probabilistic techniques in enforcing cloud-native security [6].

The general workflow of KubeGuardian is shown in Figure 1 which shows how Kubernetes API server, admission controller, policy engine, ML module and observability layer interact.

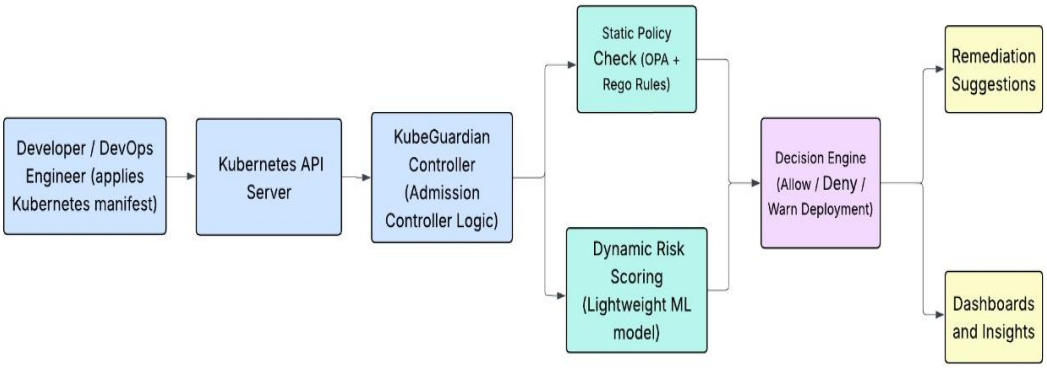


Fig. 1: Overall workflow of the KubeGuardian security enforcement framework.

3.2 System Architecture of Kube Guardian

The KubeGuardians architecture is a modular micro service based system that is deployed as a part of Kubernetes ecosystem as illustrated in figure 2. It is rationally broken down into three main layers:

1. Ingress Control Plane
2. Admission Processing Core
3. Observability and Persistence Layer

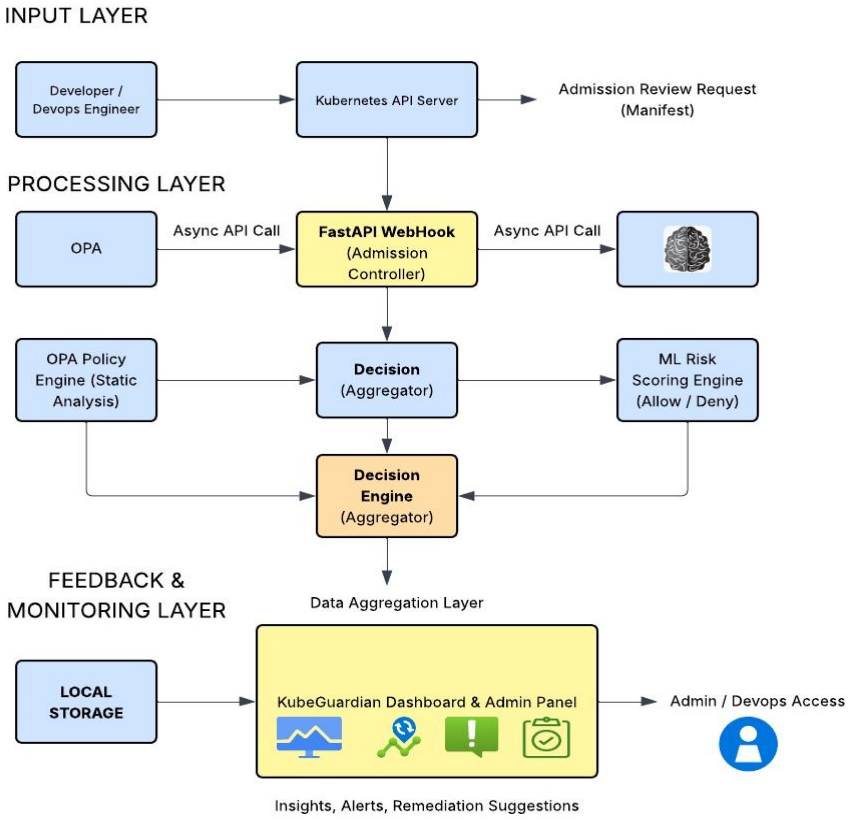


Fig. 2: Architecture of the KubeGuardian framework.

Ingress Control Plane

The Ingress control plane is the first interception point of any Kubernetes creation or modification requests of resources. When a developer sends a Kubernetes manifest, such as a Deployment, Pod, or Service, the Kubernetes API server receives the request. The KubeGuardian Validating Admission Webhook catches the request before it gets saved in the cluster datastore, etcd.

This webhook is developed based on a RESTful interface and is based on the requirements of KubernetesAdmissionReview. It is mainly used to pull out the manifest payload, normalize the configuration data and send it on to security inspection. The interception mechanism makes sure that no deployment circumvents the security validation, which is a major weakness of external scanning tools [4], [13].

3.3 Admission Processing Core

Admission Processing Core is the core analytic of KubeGuardianframework.It is composed of three closely knit subsystems:

• OPA Policy Engine
• ML Risk Scoring Engine
• Decision Engine

All these elements ensure both deterministic and probabilistic evaluation of every manifest is done.

3.3.1 OPA Policy Engine

The OPA Policy Engine implements policy rules to enforce the static security and compliance rules in the language of Rego as shown in figure 3. The module assesses Kubernetes manifests against established organizational and security best practices, such as:

- Prevention of privileged containers
- Enforcement of non-root execution
- Mandatory CPU and memory limits
- Restriction of hostPath volumes

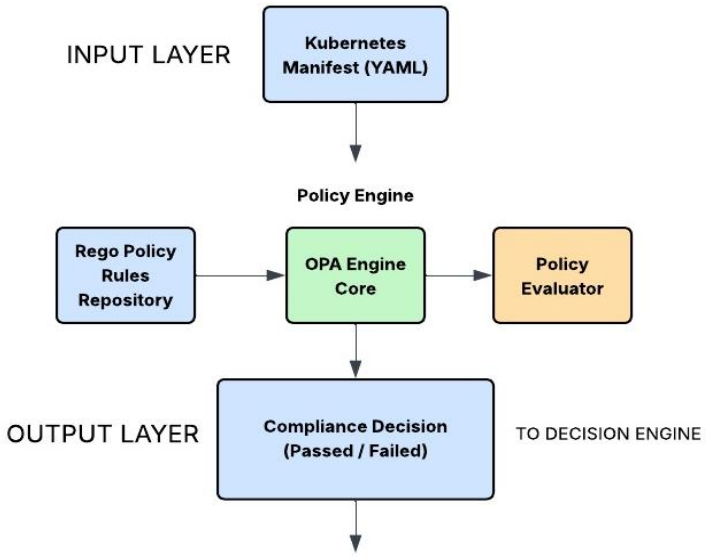


Fig. 3: Architecture of the OPA policy evaluation engine

OPA offers auditable and deterministic decisions, which is why it is appropriate to implement non-negotiable security controls [6]. Nevertheless, as mentioned in the previous literature, one cannot identify subtle or context-dependent risks with the use of a static rule-based system [13]. Such a limitation is what drives the incorporation of machine learning-based risk analysis..

3.3.2 Machine Learning–Based Risk Scoring Engine

In addition to static validation, KubeGuardian has a Machine Learning Risk Scoring Engine as shown in figure 4, which dynamically examines Kubernetes manifests. The module is based on Random Forest classifier, which was selected due to its power and interpretability, as well as its capability to work with heterogeneous sets of features [12]

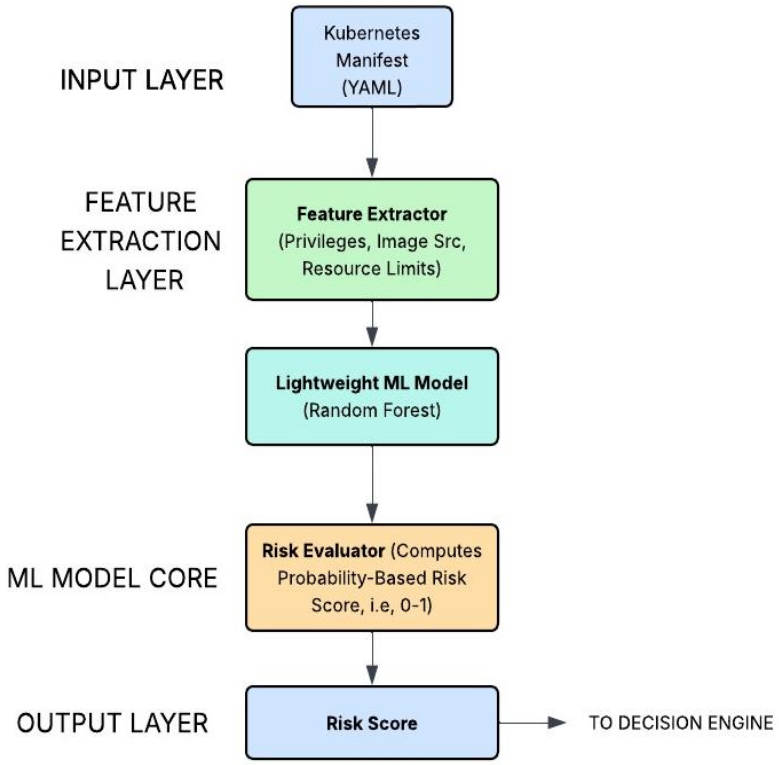


Fig. 4: ML-based Kubernetes risk evaluation system.

Feature Extraction

The manifests are converted into structured feature vectors that represent security-relevant attributes and these include:

- Presence of security contexts
- Privileged execution flags
- Resource request and limit definitions
- Exposed container ports
- Probe configurations
- Volume mount types
- Host-level access indicators

Risk Scoring

The features extracted are supplied to the trained Random Forest model that returns a risk score between 0 and 1 which is normalized. A value that goes up signifies a high possibility of security misconfiguration. In contrast to LLM-based solutions like GenKubeSec [9] and LLMSecConfig [16], this methods is focused on low-latency inference and interpretability; thus, it can be used in real-time admission control.

3.3.3 Decision Engine

The Decision Engine combines the results of both OPA Policy Engine and the ML Risk Scoring Engine. It gives one of the following decisions, depending on set thresholds and degrees of severity:

- **Allow**: Manifest is compliant and low-risk
- **Warn**: Manifest passes policies but exhibits elevated risk
- **Deny**: Manifest violates mandatory policies or exceeds risk thresholds

The hybrid decision-making method corresponds to the body of research that suggests prioritization of risk enforcement and contextual prioritization of risk [1], [15].

The ML Risk Scoring Engine produces a normalized risk score in the range [0,1]. Admission decisions are derived using the following thresholds:

• Allow: Risk Score < 0.40 and no critical OPA violations.
• Warn: 0.40 ≤ Risk Score < 0.75 and no critical OPA violations.
• Deny: Risk Score ≥ 0.75 or any critical OPA policy violation.

The threshold values were determined empirically through validation-set optimization, maximizing the F1-score while minimizing false negatives associated with insecure deployments.

3.4 Admission Controller Webhook Integration

The Admission Controller Webhook as shown in figure 5. is the working interface between the Kubernetes API server and the analytical engines.

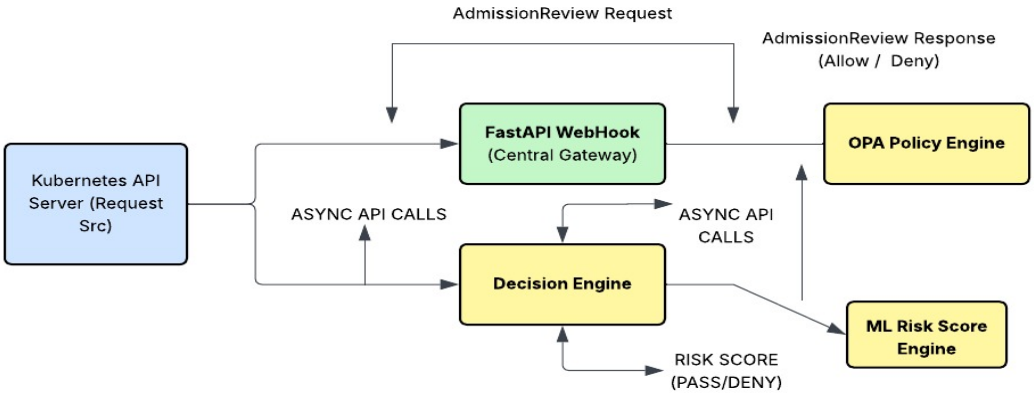


Fig. 5: Architecture of the admission controller webhook.

When a request to deploy is received, the webhook:

1. Submits the manifest to the OPA Policy Engine.
2. Starts with the extractions of features and ML-risks scoring.
3. Invokes the Decision Engine
4. Sends an admission response (Allow/Deny) to the API server.
5. Forwards validation responses to the observability layer.

This design guarantees synchronous validation, but does not exceed the requirement of admission latency of Kubernetes.The "Async API Calls" shown in the architecture diagrams represent internal service communication and not asynchronous admission decisions. Consequently, all deployment requests undergo complete validation before admission, preserving Kubernetes admission-control semantics.

3.5 Observability and Admin Dashboard

Admin Dashboard as shown in figure 6, allows viewing all the risk judgments and admissions.

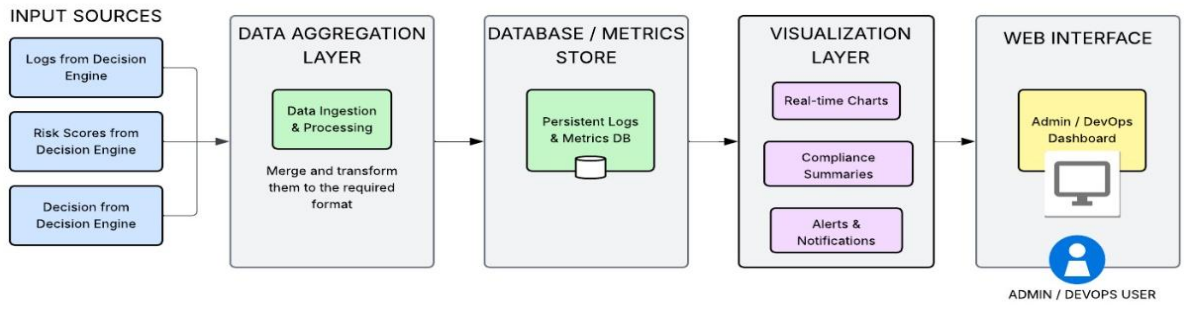


Fig. 6: Architecture of the KubeGuardian admin dashboard.

The dashboard visualizes:

- Validation outcomes
- Risk score distributions
- Policy violation statistics
- Historical trends and alerts

The usability problem found in the existing studies, including the lack of actionable feedback to use the security tools [4], [10], is also addressed by the layer of observability.

3.6 Dataset Preparation and Pre-processing

The preparation of the datasets used to train and evaluate the ML Risk Scoring Engine is taken care of by a special module. Figure 7 illustrates the dataset generation and pre-processing workflow.

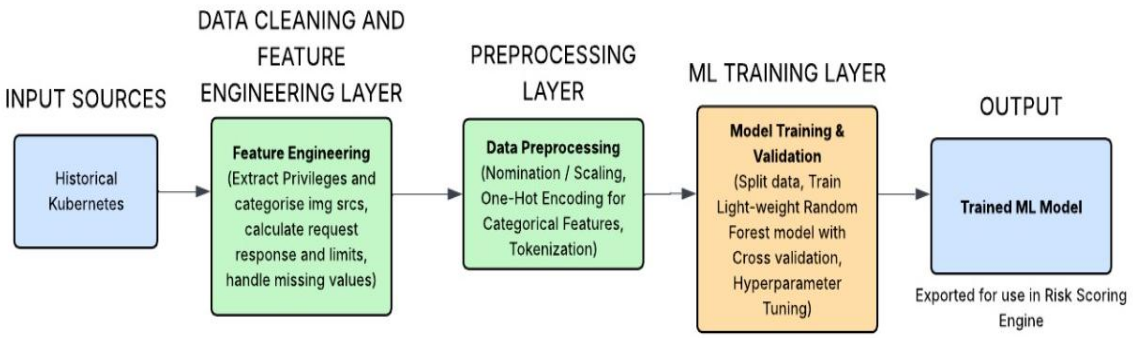


Fig. 7: Dataset preparation and pre-processing pipeline.

The dataset used for training and evaluating the ML Risk Scoring Engine consisted of 10,000 Kubernetes manifests collected from public repositories, Kubernetes benchmark examples, and synthetically generated configurations. The dataset contained 5,200 secure manifests and 4,800 insecure manifests, ensuring a balanced class distribution.

To avoid circular dependency between policy enforcement and machine learning prediction, manifest labels were not derived solely from OPA policy decisions. Instead, labels were assigned using a combination of :
• CIS Kubernetes Benchmark recommendations,
• Security expert review,
• Results from Checkov and kube-bench scans,
• Manual validation of high-risk configurations.

The dataset was partitioned using a stratified 70:15:15 split for training, validation, and testing respectively. Feature extraction included security-context settings, privilege escalation flags, host-network usage, hostPath mounts, resource constraints, exposed ports, probe configurations, and RBAC-related permissions.Five-fold cross-validation was performed during model development to ensure generalization and reduce overfitting [12].

3.7 Summary of the Proposed Methodology

The suggested approach facilitates proactive and real-time Kubernetes security implementation through integrating deterministic policy tests and real-time risk scoring in the admission control lifecycle. Keeping contextual intelligence without much inference of heavy LLM, KubeGuardian strikes the balance of security rigor, scalability, and operational performance, overcoming the major setbacks of current solutions [9], [13], [16].

4. Result and Discussion

Researchers This section will provide a detailed assessment of the suggested KubeGuardian framework in performance and efficiency of the framework modules. The findings are discussed to illustrate how the framework can identify the presence of Kubernetes misconfigurations, apply security policies dynamically, and give useful risk data with limited operational costs. Synthesized and real-world Kubernetes manifests were used to evaluate the experimental effectiveness of the policy, along with both the manual and machine-learning-based

risk assessment.

The assessment is divided into two primary sections:
(i) Module-wise results, which highlight each system component's performance and behavior, and
(ii) Machine learning risk scoring model performance, i.e. standard classification metrics.

4.1 Module-wise Results

OPA Policy Engine

The OPA Policy Engine was tested as having the capacity to impose deterministic security policies on Kubernetes manifests during admission time. An extensive list of Rego policies was also enforced to identify typical but serious misconfigurations, such as privileged containers, the lack of security contexts, uncontrolled access to a host, lack of resource constraints, and unsafe volume mounts.

The OPA engine was able to undertake the task of static validation of incoming Kubernetes manifests and produce policy decisions which are either Allow, Warn or Deny as seen in Figure 8.

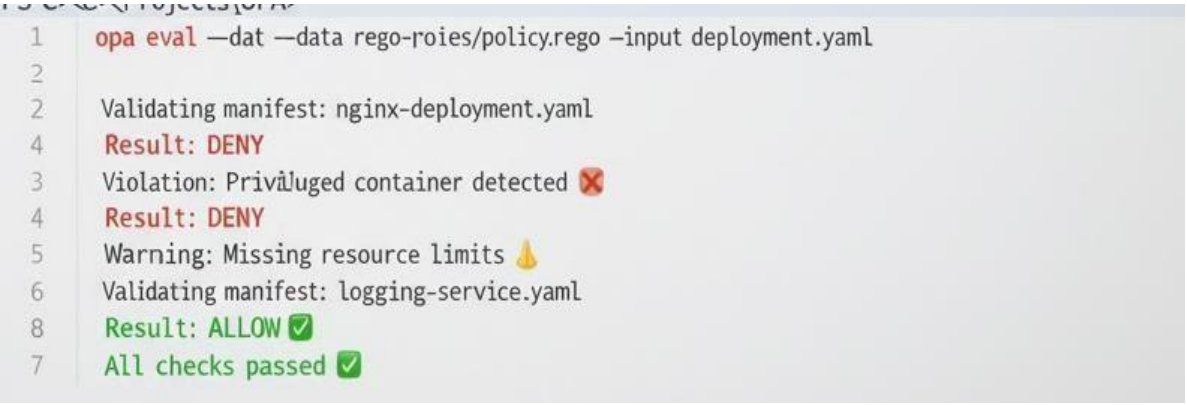


Fig. 8: OPA Policy Engine performing static validation of Kubernetes manifests.

The experiments conducted prove that manifests with privileged containers or incomplete non-root execution constraints were always rejected at admission time. Partial compliance, i.e., manifests that lacked resource limits but contained no critical security violations were marked with a warning. Manifold conforming manifolds were permitted immediately.

These results attest the fact that OPA offers credible and trusted security control enforcement of non-negotiable security controls. Operational wise, the OPA engine showed insignificant evaluation latency, and hence it could be applied to real time admission control. Nonetheless, the conventional validation was not adequate to identify subtle or context-specific risks, like combinations of what appear to be harmless configurations that compound attack surface. This weakness encourages risk scoring dynamism.

Machine Learning Risk Scoring Module

The ML Risk Scoring unit is used to supplement the policy enforcement that is not dynamic by offering dynamic, probabilistic analysis of Kubernetes manifests. The module is based on a Random Forest classifier, trained on a dataset based on both actual and artificial Kubernetes configurations.

The identified features extraction pipeline revealed security-sensitive features like security context presence, container privilege flags, resource constraints, exposed ports, probe definitions, and indicators of host-level access. These characteristics were transferred to the trained model to produce a normalized risk score of 0 to 1.

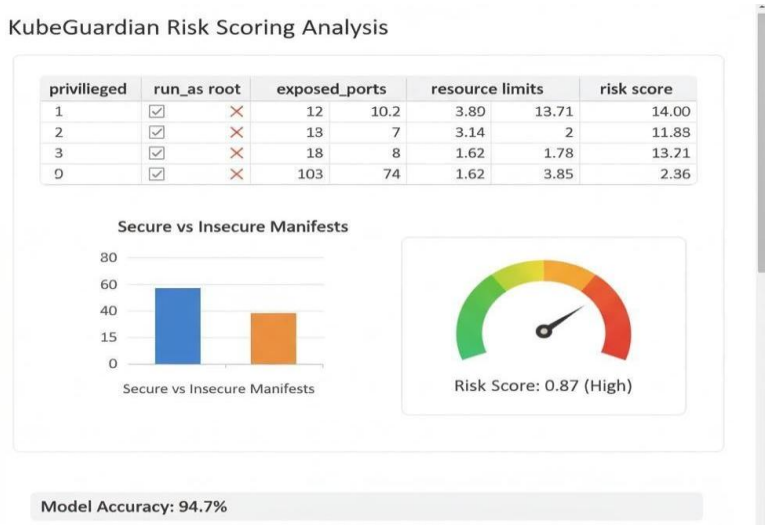


Fig. 9: ML Risk Scoring module showing extracted features and predicted risk score.

The unsafe or high-risk configurations in the manifests were also associated with high risk scores (0.75 and above) and compliant manifests scored much lower (0.25 and below) as illustrated in Figure 9. The model was effective in distinguishing between secure and insecure set up, even when the violations occurred without the help of any static policies.

One of the benefits of the Random Forest model is that it is interpretable. The analysis of the feature importance demonstrated that privileged execution, use of host network, and lack of resource limits were some of the most significant factors that contributed to high-risk predictions. This interpretability makes the decisions of the model more trustworthy and facilitates remediation.

Admission Controller Webhook Performance

The admission controller Webhook is a FastAPI-based server that is the interface between the Kubernetes API server and the validation engines. Its performance was also tested on the parameters of correctness, responsiveness, and reliability when it is subjected to repeated admission requests.

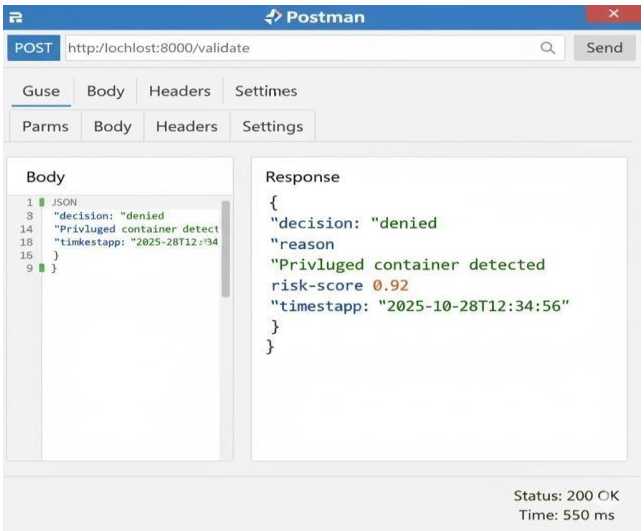


Fig. 10: FastAPIwebhook validation showing real-time admission control decision

The findings show that the webhook has been able to intercept deployment requests, send manifests to the OPA Policy Engine and ML Risk Scoring module as illustrated in figure 10 , combine their results and provide both modules with unified admission decisions in real time. Test cases that were run with the help of Postman and simulated Kubernetes deployments showed correct and consistent responses.

In the case of insecure manifests, the webhook responded with a Deny command with detailed policy violation messages and risk scores. Secure manifests were accepted with a latency of no notice. Notably, the webhook complied with Kubernetes admission time limits and did not interrupt cluster operations. This confirms the practicability of KubeGuardian implementation in production-scale Kubernetes setups

Admin Dashboard and Observability Layer

Admin Dashboard is essential in converting raw validation outputs into meaningful security insights. It can take admission decisions, risk scores, and policy violation metadata and provides them in a graphical interface that is easy to use.

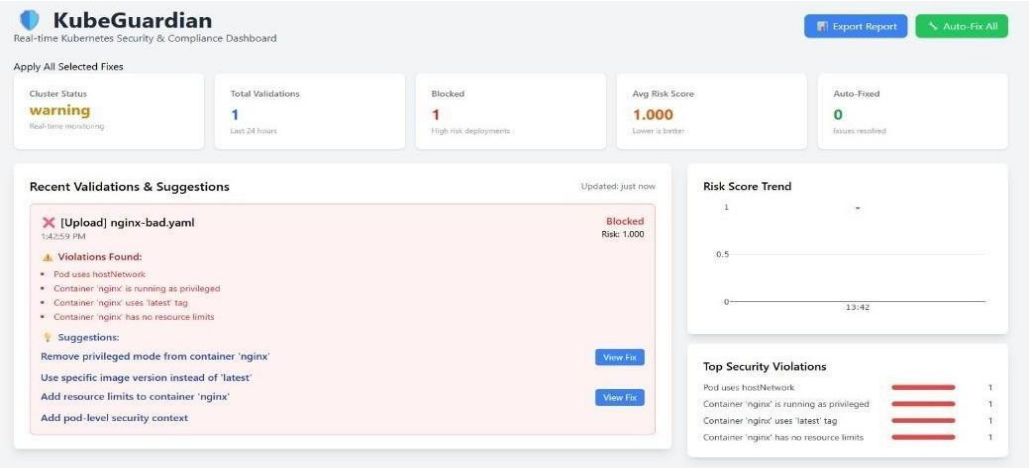


Fig. 11: User interface of the KubeGuardian admin dashboard.

the dashboard provides real-time visualization as shown in figure 11 of:

- Total validated manifests
- Allowed vs. denied deployments
- Risk score distributions
- Frequently violated policies
- Security posture temporal trends.

The dashboard will allow the administrators to see the patterns of misconfigurations, track compliance patterns, and prioritize the remediation process. Contrary to the conventional methods of static tools, which generate detached reports, the dashboard provided by KubeGuardian can have a high level of continuous observability, which is far much better in matters of operational awareness and governance.

Dataset Preparation and Feature Extraction Results

The Dataset Preparation and Preprocessing module was checked in order to determine the accuracy and reflectivity of the training data applied to the ML Risk Scoring model.

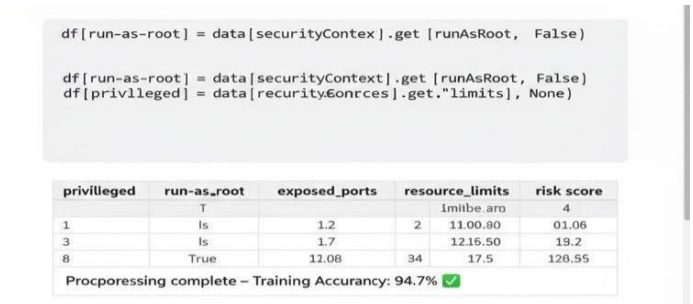


Fig. 12: Dataset preparation and feature extraction process for ML risk model

The dataset as illustrated in figure 12 had a balancing proportion of compliant and non-compliant manifests and controlled variation of security attributes of importance. Normalization and encoding of features made them

compatible with the Random Forest model and avoided the bias with high-magnitude features.

4.2 Performance Evaluation of the ML Risk Scoring Model

The performance of the ML Risk Scoring module was measured quantitatively with the help of standard classification measures. The summary of the performance of the Random Forest classifier is presented in Table 4.1.

Table 4.1. Performance Evaluation of ML Risk Scoring Model.

METRICS	SCORE (%)
Accuracy Score	94.6
Precision Score	92.1
Recall	93.5
F1-Score	92.8

The results seem to support that the model was very predictive in all measures and hence can be used to be very reliable in distinguishing between secure and insecure settings. False positives are minimalized with high precision, and false negatives are minimized with high recall. The administrators need not be fatigued by alerts and critical misconfigurations are not neglected. The bar chart illustrates the performance of the Random Forest classifier used in the ML Risk Scoring module, as shown in **Figure 13**. The model achieves a high accuracy of **94.6%**, indicating reliable overall prediction capability. A precision of **92.1%** reflects a low false-positive rate, ensuring that secure Kubernetes manifests are not incorrectly flagged as insecure. The recall value of **93.5%** demonstrates the model's effectiveness in identifying insecure configurations. The balanced **F1-score of 92.8%** confirms consistent performance across both precision and recall, thereby validating the robustness and reliability of the proposed ML-based risk assessment approach.

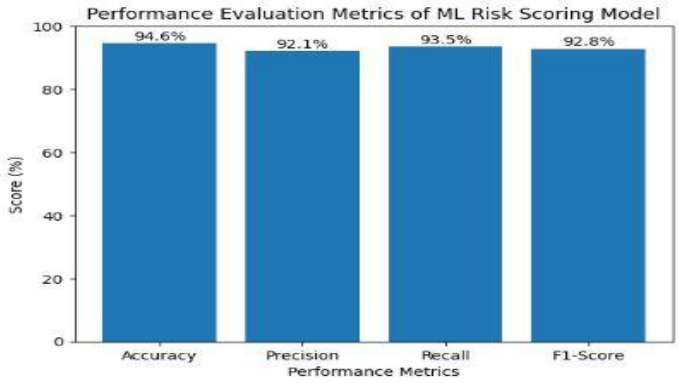


Fig. 13: Performance Evaluation Metrics of ML Risk Scoring Model

The ML-based risk scoring showed a significant improvement in identifying subtle misconfigurations that do not necessarily breach explicit policies, but add to the increased security risk compared to the traditional methods of risk scoring which are based solely on statistics. Lightweight Random Forest model, in contrast to the solutions based on LLM, achieved this without adding much computational overhead and inference latency.

4.3 Ablation Study

To evaluate the contribution of individual framework components, three configurations were compared:

1. OPA Only
2. ML Risk Scoring Only
3. Combined KubeGuardian Framework

Table 4.2: Detection Accuracy and False Positive Rate

Configuration	Detection Accuracy (%)	False Positive Rate (%)
OPA Only	87.3	8.9
ML Only	90.8	7.5
OPA + ML (KubeGuardian)	94.6	4.8

The results indicate that OPA effectively detects explicit policy violations, whereas the ML module identifies subtle configuration risks that may not violate predefined policies. The combined framework achieved the highest detection accuracy and lowest false-positive rate, validating the effectiveness of the hybrid approach as shown in table:4.2.

4.4 Admission Latency Analysis

Admission latency was evaluated using 1,000 deployment requests of varying manifest complexity.

Table 4.3: Metric

Metric	Latency (ms)
Mean Latency	41.8
Median Latency	38.5
95th Percentile	67.3
99th Percentile	92.1

The results demonstrate that KubeGuardian satisfies Kubernetes admission-control timing requirements while providing both policy enforcement and risk assessment as shown in the table: 4.3. The lightweight Random Forest model contributed less than 8 ms of average inference overhead.

4. Conclusion

The paper has presented KubeGuardian, a real-time and intelligent Kubernetes security framework that can be implemented to address the rising associative vulnerabilities in cloud-native environments that are configuration-based. Though Kubernetes enables scalability and flexibility of orchestrating applications, it has a large attack surface, as it is based on a declarative model of configuration, which comprises of YAML. One of the most prevalent results of security breach and operational failures in Kubernetes clusters are privileged containers, lack of security contexts, unreasonable resource assignments, and unreasonable access control.

Unlike conventional external or post-deployment tools of conventional analysis, KubeGuardian is integrated into Kubernetes control plane via a Validating Admission Controller, and proactive security policy enforcement can be exercised before workloads are accepted into the cluster. Deterministic validation using an Open Policy Agent (OPA) holds that organizational and compliance policy are followed to the letter which is auditable and reproducible governance which adheres to the tenets of the policy-as-code. In response to the shortcoming of rule-based systems, KubeGuardian has an engine of risk scoring that relies on machine learning and that can dynamically evaluate security risks in the context of lightweight and understandable models. This mixed approach facilitates the possibility of identifying fine misconfigurations that might remain unnoticed under pure policies but maintains low-latency performance desirable to real-time admission processes.

Through experimental testing, it was found that KubeGuardian is viable enough to identify and prevent insecure deployments, assign a meaning risk score, and practical remediation advice with the help of an interactive administrative interface. In comparison to the GenKubeSec and LLMSecConfig models, both of which are based on the LLM and yield high quality semantic reasoning but at high computing cost KubeGuardian offers a reasonable tradeoff between intelligence, interpretability and scalability. The dashboard-driven observability also results in the enhanced comprehension of the operations as enabling one to trace the pattern of compliance and repetition of security violations with the purpose of addressing the identified gaps in usability in the existing tools.

In total, KubeGuardian helps to develop Kubernetes security management through a unified framework, which combines operational visibility with adaptive risk assessment and enforcement in real time. Its modular and cloud based design allows it to support up to date DevSecOps and GitOps, and increasing its continuous adherence to dynamical deployment environments.

References

1. Balduzzi, M., Zaddach, J., Balzarotti, D., et al., “A Security Analysis of Kubernetes Deployments,” Proceedings of the ACM Cloud Computing Security Workshop, 2022.

2. Bufalino, J., et al., “Inside Job: Characterizing Kubernetes Network Misconfigurations,” Proceedings of the ACM SIGSAC Conference on Computer and Communications Security,2022.

3. Kubernetes Documentation, “Kubernetes Architecture and Components,” [Online]. Available: https://kubernetes.io/docs/concepts/. Accessed: 2024.

4. Chen, Y., et al., “An Empirical Study on Helm Chart Vulnerabilities and Fixability,” IEEE Transactions on Software Engineering, 2022.

5. Checkov Documentation, “Infrastructure as Code Static Analysis,” [Online]. Available: https://www.checkov.io. Accessed: 2024.

6. Di Martino, B., et al., “Policy-as-Code for Security and Privacy in Cloud-Native Systems,” Future Generation Computer Systems,2021.

7. Gatekeeper Documentation, “OPA Gatekeeper for Kubernetes,” [Online]. Available: https://open-policy-agent.github.io/gatekeeper/. Accessed: 2024.

8. Kubernetes Security Best Practices, “Common Kubernetes Misconfigurations,” [Online]. Available: https://kubernetes.io/docs/concepts/security/. Accessed: 2024.

9. Gandikota, Hari Prasad, Abirami S, and S Kumar M. "CT scan pancreatic cancer segmentation and classification using deep learning and the tunicate swarm algorithm." Plos one 18.11 (2023): e0292785.

10. Kumar, M. S, et al. "Simulation of the electrical control unit (ECU) in automated electric vehicles for reliability and safety using on-board sensors and Internet of Things." 2023 Fifth International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE, 2023..

11. Narayanan, L., et al., “A Large-Scale Study of Kubernetes Manifest Misconfigurations,” Journal of Internet Services and Information Security,2023.

12. Racharla, M., et al., “Automated Bug Preprocessing and Classification Using NLP and Machine Learning,” International Journal of Software Engineering and Knowledge Engineering,2021.

13. Bikku, Thulasi, et al. "Healthcare biclustering of predictive gene expression using LSTM based support vector machine." Informing Science 28 (2025): 12.

14. Girinath, S., et al. "Real-time identification of medicinal plants using deep learning techniques." 2024 International Conference on Trends in Quantum Computing and Emerging Business Technologies. IEEE, 2024.

15. Tayouri, D., et al., “CORAL: Logic-Based Risk Assessment for Containerized Environments,” ACM Transactions on Privacy and Security, 2022.

16. Ye, Z., et al., “LLMSecConfig: Automated Detection and Repair of Container Configuration Errors Using Large Language Models,” Proceedings of the IEEE International Conference on Software Engineering, 2024.

17. Al-Refai, O., & Al Atawneh, S. J. (2026). Sharp criteria for starlike and convex univalent functions. *Results in Nonlinear Analysis*, 9(2), 37-46.