National College of Ireland

Pre-Deployment CIS Risk Assessment and Mitigation for Helm Charts Using AI

MSc Research Project
Cloud Computing

Ajay Panwar
Student ID: 23270411

School of Computing
National College of Ireland

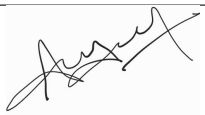
Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ajay Panwar
Student ID:	23270411
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	11/08/2025
Project Title:	Pre-Deployment CIS Risk Assessment and Mitigation for Helm Charts Using AI
Word Count:	4858
Page Count:	18

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:	
Date:	11th September 2025

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):	

Pre-Deployment CIS Risk Assessment and Mitigation for Helm Charts Using AI

Ajay Panwar
23270411

Abstract

The widespread adoption of Kubernetes and Helm charts has accelerated application deployment but introduced significant security risks from misconfigurations that violate Center for Internet Security (CIS) Benchmarks. Given that existing scanners overwhelm developers with unprioritized alerts and lack CIS-specific risk scoring, there is a clear need for a proactive, pre-deployment assessment. This paper presents a novel, lightweight, rule-based AI framework that parses rendered Helm templates to analyze Kubernetes manifests against 25 CIS controls. A custom risk model quantifies non-compliance with weighted severity scoring to generate actionable risk scores (0–100) and precise remediation guidance. Empirical validation on 25 popular Helm charts demonstrated a 96% remediation success rate, reduced aggregate risk from 750 to 35 points (95% reduction), and an average processing time of 0.025 seconds per chart. Comparative evaluation with Kube-Linter and post-deployment checks via Kube-bench confirmed the framework's practical applicability. In 25 charts tested, Kube-bench failures dropped 64% (from 312 to 112), following our pre-deployment fixes. In practice, this shift-left paradigm empowers developers to address misconfigurations before production, reducing potential security incidents. Remaining challenges include extending rule coverage to network policies and improving runtime context awareness.

1 Introduction

This section introduces the research by providing the necessary context and motivation for the work undertaken. It establishes the background of Kubernetes and Helm security complexities, highlights critical gaps in current security practices, and presents the central research question and objectives.

1.1 Background

Kubernetes is the de-facto container orchestration standard, but its myriad configuration options expand the attack surface when defined via Infrastructure as Code (IaC). A single misconfigured `YAML` setting can compromise an entire cluster.

Helm charts simplify deployments yet introduce supply-chain risks: many public charts ship with insecure defaults or outdated images. Existing scanners (static pre-deployment or runtime) overwhelm developers with unprioritized alerts. This work shifts CIS Benchmark checks left, producing a single risk score plus precise remediation before deployment.

1.2 Importance and Contribution

Public Helm charts create an unrecognized security gap. Development teams deploy charts presuming they meet baseline security standards, but these baselines are often not realized. This research establishes a practical framework combining IaC security, Helm package management, and developer-centric tooling.

The primary contribution is a novel system that evaluates and mitigates security risks before deployment. The framework addresses alert fatigue by consolidating CIS benchmark findings into a single, quantifiable Overall Risk Score and clear Risk Category. This transforms overwhelming security data into actionable signals, enabling teams to prioritize remediation effectively. Beyond the core framework, this research demonstrates extensibility to emerging security standards and cross-platform applicability.

1.3 Research Question and Objectives

1.3.1 Research Question

How can a lightweight, AI-driven framework effectively evaluate CIS benchmark compliance, quantify security risks, and guide mitigation for Helm charts in a pre-deployment context?

1.3.2 Research Objectives

- **RO1:** Develop a pre-deployment analysis pipeline that renders Helm charts, generates CIS benchmark security findings, supports extensible compliance frameworks, leverage a lightweight AI model for actionable risk scoring, and enables guided mitigation with precise remediation instructions and automated secure configuration templates.
- **RO2:** Empirically evaluate the framework's effectiveness across 25 diverse Helm charts, using before-and-after analysis to demonstrate measurable risk reduction and cross-platform applicability.

1.4 Limitations

This research focuses on CIS Benchmark compliance for Helm charts and does not address broader Kubernetes security concerns such as network policies or RBAC configurations. The evaluation is limited to 25 popular charts. The AI model is rule-based rather than machine learning-based, limiting adaptability to new threat patterns without manual rule updates.

1.5 Report Structure

The remainder of this paper is structured as follows:

- **Section 2** reviews related work on Kubernetes security, Helm chart vulnerabilities, and automated risk analysis tools.
- **Section 3** outlines the research methodology, including the design principles and evaluation strategy.
- **Section 4** presents the architecture and specification of the proposed framework.

– **Section 5** details the implementation of the toolchain, including the analysis pipeline and AI-based scoring system.

– **Section 6** evaluates the framework's effectiveness using real-world Helm charts and CIS benchmark compliance.

– **Section 7** concludes the paper with key findings, limitations, and suggestions for future work.

2 Related Work

2.1 Introduction

This section provides a comprehensive review of existing knowledge pertinent to Kubernetes security, focusing on challenges posed by Helm chart deployments. The review is structured to build a foundational understanding of the problem domain, starting from IaC security and narrowing to pre-deployment analysis, compliance standards, and the emerging role of Artificial Intelligence (AI) in automating security. The primary objective is to survey the state-of-the-art, critically evaluate existing methodologies, and identify the specific research gap this paper addresses.

2.2 The Security Landscape of Kubernetes and Infrastructure as Code (IaC)

Kubernetes has emerged as the standard for container orchestration, bringing immense agility to organizations, yet such flexibility comes at a price: complexity. Islam Shamim et al. (2020) acknowledged that a standard Kubernetes environment has multiple connected components with many configurations that create a complex and broad attack surface. Infrastructure as Code manages that complexity through declarative machine-readable files for infrastructure versioning and deployment. However, IaC introduces new classes of vulnerabilities: misconfigurations. Rahman et al. (2019) identified seven "security smells" in IaC scripts such as hard-coded secrets and overly permissive policies. This was reinforced by Rahman (2018b) claiming such defects are common in DevOps.

Recent empirical work by Rahman et al. (2023) conducted around thousands of open-source Kubernetes manifest documents found security misconfigurations exist frequently. Bose et al. (2021) confirmed that many security defects are under-reported, meaning the issue is larger than publicly visible. Additionally, Wu et al. (2020) demonstrated that container-level security issues extend beyond Kubernetes manifests to Dockerfile configurations, highlighting the pervasive nature of misconfigurations across the containerization stack. Rahman et al. (2021) extended the concept of security 'smells' to Ansible and Chef scripts, identifying common misconfigurations—such as hard-coded credentials and missing input validation—that parallel issues found in Kubernetes manifests and underscore the broader challenge of IaC security across toolchains.

While these studies comprehensively catalogue IaC misconfigurations, they stop short of offering unified severity prioritization, leaving a gap that rule-based scoring can fill.

2.3 Helm Charts as a Primary Vector for Risk

To manage deployment complexity, the ecosystem has adopted Helm, the platform's official package manager. Public repositories like Artifact Hub host vast ecosystems of community-contributed charts, enabling single-command deployments of complex software. This reliance on third-party charts introduces significant supply chain security risks. Zerouali et al. (2023) analyzed charts on Artifact Hub and found many contained outdated container images with known vulnerabilities, accumulating "technical lag." Liu et al. (2020) extended this analysis to container registries, demonstrating widespread security issues in base images used by these charts.

Beyond outdated dependencies, default configurations within Helm charts are often insecure. Minna, Massacci and Tuma (2024) found high prevalence of misconfigurations, such as containers running as root and disabled security policies. Blaise and Rebecchi (2022) demonstrated how these misconfigurations could be modeled as attack graphs, showing concrete pathways for cluster compromise. Gajananan et al. (2021) addressed runtime protection but highlighted the need for pre-deployment solutions. Minna, Blaise, Massacci and Tuma (2024) evaluated five static analyzers and introduced chart-specific repair transforms, reducing manual patch effort by 45%. Kamieniarz and Mazurczyk (2024) compared multiple Kubernetes deployment methods, reporting that Helm-based deployments scored 20 percent lower on security metrics due to insecure defaults, thereby motivating targeted pre-deployment analysis for chart configurations.

Although these works underscore Helm's risks, they lack prescriptive remediation guidance, indicating the necessity for our AI-driven mitigation instructions. Chen et al. (2023) conducted a large-scale mixed-methods study on 11,035 Helm charts, finding that chart complexity correlates strongly with fixable vulnerabilities and demonstrating the low adoption of automated mitigation strategies. Suresh (2024) demonstrated in the serverless domain that integrating IaC-specific threat modeling into deployment pipelines can detect 100% of intentionally injected vulnerabilities and reduce provisioning times by over 80%, illustrating the effectiveness of pre-deployment IaC security assessment beyond container orchestration contexts. Shamim (2021) proposed an automated mitigation framework that detects and fixes security-best-practice violations in Kubernetes manifests, reducing exploit paths by remediating common misconfigurations before deployment.

2.4 Establishing Security Baselines: CIS Benchmarks and Compliance

Industry bodies have developed standardized security guidelines in response to securing complex IT systems. The CIS Benchmarks are globally recognized best practices. Sedano and Salman (2021) demonstrated these benchmarks provide prescriptive hardening guidance.

The CIS Kubernetes Benchmark covers master and worker node components and pod security policies. Cauli et al. (2021) further showed that encoding IaC templates into description logic models allows formal pre-deployment verification of compliance and inter-resource dataflows, enabling automated detection of higher-order security issues. Adherence is critical for establishing strong security baselines and regulatory compliance. However, manual auditing is impractical and error-prone. Bin Khalid (2025) explored automated compliance assessment methods for GKE clusters using tools like Kube-bench.

Nerella and Puvvada (2024) provided comprehensive analysis of enterprise Kubernetes security gaps, underscoring the necessity of automated tools for CIS compliance at scale.

These contributions validate automated compliance assessments but focus on live clusters, leaving pre-deployment scanning underexplored.

Table 1: Comparative Analysis of Key Papers versus the Research Conducted

Author(s)	**Pre-deployment Analysis**	**Helm Chart Specific**	**CIS Benchmark Compliance**	**Lightweight AI Approach**	**Risk Score Generation**	**Automated Mitigation Suggestions**
Zerouali et al. (2023)	✓	✓	✗	✗	✗	✗
Minna, Massacci and Tuma (2024)	✓	✓	✗	✗	✗	✓
Blaise and Rebecchi (2022)	✗	✓	✗	✗	✗	✓
Malul et al. (2024)	✓	✗	✗	✗	✗	✓
Gajananan et al. (2021)	✗	✓	✗	✗	✗	✗
Bin Khalid (2025)	✗	✗	✓	✗	✓	✗
Kapetanidou et al. (2025)	✗	✗	✓	✗	✗	✗
Rahman et al. (2023)	✓	✗	✗	✗	✗	✗
Sedano and Salman (2021)	✗	✗	✓	✗	✗	✗
Chaganti (2025)	✗	✗	✗	✓	✗	✓
Our Proposed Research	✓	✓	✓	✓	✓	✓

2.5 State-of-the-Art in Kubernetes Security Scanning

A diverse ecosystem of scanning tools has emerged, categorized into pre-deployment (static analysis) and post-deployment (runtime analysis) solutions. Kapetanidou et al. (2025) provides useful evaluation of several popular tools, offering insights into their capabilities and limitations. However, significant gaps exist in providing actionable, prioritized, benchmark-aligned feedback for Helm chart deployments. While Kube-bench is trusted for CIS compliance, it only works against running clusters. Pre-deployment scanners like Checkov, Polaris, and Datree excel at identifying misconfigurations but use proprietary or generalized best practices rather than comprehensive CIS compliance checks.

The absence of consolidated risk scoring and Helm-specific CIS checks highlights a clear opportunity for our specialized framework. Recent work by Rahman (2018a) on IaC anti-patterns identified seven common Helm chart security smells that persist across 1,500 real-world charts, illustrating the need for specialized chart checks. The comparative analysis in Table 1 highlights the gaps in previous works suggesting the need for proposed

solution.

2.6 Synthesis and Identification of the Research Gap

The literature review clarifies several points: Kubernetes and Helm introduce complexity and risk; misconfigurations are common and serious security problems; CIS Benchmarks provide strong security standards but require automation; existing tools lack prioritization and benchmark alignment in pre-deployment analysis; and AI offers powerful emerging automation options.

Synthesizing these points reveals a clear research gap: the need for an integrated, pre-deployment framework that specifically targets Helm charts, validates them against the official CIS Kubernetes Benchmark, and provides consolidated risk scoring with guided remediation—capabilities absent from existing work.

3 Methodology

This section details the systematic methodology designed to answer the research question through rigorous experimentation and validation. The approach follows a controlled, comparative experimental design to evaluate the effectiveness of a custom AI-driven framework for pre-deployment security analysis of `Helm` charts, with validation through post-deployment verification.

3.1 Research Design and Approach

This research employs a quantitative experimental methodology with a before-and-after comparative analysis design. The study utilizes 25 real-world Helm charts as the primary dataset, applying a systematic intervention (AI-driven security analysis and mitigation) and measuring resulting security posture improvements. The experimental design includes control mechanisms through established industry tools to ensure validity and enable comparative analysis.

Alternative methodologies were considered, including machine learning-based approaches and post-deployment-only analysis. However, a rule-based AI system was selected for transparency and deterministic behavior, while pre-deployment analysis was chosen to enable proactive security improvements before production deployment.

3.2 Data Collection

3.2.1 Sample Selection and Infrastructure Setup

The research utilized purposive sampling to select 25 popular `Helm` charts (list included in the configuration manual) from established repositories (`Bitnami`, `Ingress-NGINX`, `JFrog`) based on popularity metrics, application diversity, and maintenance status. Charts were version-pinned to ensure reproducibility.

Infrastructure comprised Google Kubernetes Engine (GKE) for validation testing, with Windows Subsystem for Linux (WSL) providing the development environment. The cloud infrastructure was selected to represent enterprise deployment scenarios while providing standardized testing conditions.

3.2.2 Tools and Equipment

Key tools used in this study included `KubeLinter` v0.6.6 for baseline static analysis, `Kubescape` v3.0.34 for CIS benchmark compliance scanning, and `Kube-bench` for authoritative post-deployment CIS validation. Custom tool development was implemented using `Python`, leveraging libraries such as `PyYAML` and `argparse`. `Helm` v3.x was employed for chart templating and deployment throughout the evaluation process. Tool selection was based on industry adoption, CIS benchmark alignment, and integration capabilities with Continuous Integration and Continuous Delivery/Deployment (CI/CD) pipelines.

3.3 Data Processing and Analysis

3.3.1 Data Processing Pipeline

Data processing followed a standardized four-step pipeline designed to convert `Helm` charts into analyzable security data. First, charts were rendered into static `Kubernetes` manifests using the `helm template` command. Next, a custom `Python` script (`helm_parser.py`) extracted security-relevant fields such as privilege settings, capabilities, and access controls from the generated `YAML` files. These extracted configurations were then evaluated using a rule-based engine aligned with the CIS `Kubernetes` Benchmark. The final step involved quantifying risk: a weighted scoring algorithm produced normalized risk scores ranging from 0 to 100, alongside categorical classifications (e.g., `Low`, `Medium`, `High`, `Critical`) based on the severity and number of violations.

3.3.2 Measurement Techniques

Quantitative evaluation relied on three primary metrics. First, risk was assessed using a weighted severity scoring system, where each `Critical` issue contributed 30 points, `High`-severity issues 15 points, and `Medium`-severity issues 5 points. Second, compliance violations were tallied and categorized by severity, enabling comparative analysis across different charts and configurations. Lastly, processing time was measured to evaluate the performance and efficiency of the analysis pipeline, ensuring the framework remains lightweight enough for integration into continuous delivery environments. Qualitative measurements captured remediation guidance effectiveness and framework usability through structured analysis of generated recommendations.

3.4 Experimental Intervention and Validation

The experimental intervention applied a three-step process to each chart: (1) initial risk assessment using the custom framework, (2) guided mitigation application based on generated recommendations, and (3) re-assessment validation to measure improvements. Control mechanisms included parallel analysis using `KubeLinter` for baseline comparison and post-deployment validation using `Kube-bench` in live `GKE` clusters to verify that pre-deployment improvements translated to operational security enhancements. Statistical analysis employed descriptive statistics to characterize security improvements and comparative analysis to evaluate framework performance against established tools. Before-and-after comparisons enabled precise measurement of intervention effectiveness across the complete dataset.

3.5 Reproducibility and Validity

Internal validity was maintained through standardized procedures, consistent tool versions, and controlled environment configurations. All experiments utilized identical infrastructure and analysis parameters to eliminate confounding variables. External validity was supported through diverse chart selection representing multiple application categories and organizational sources, ensuring findings generalize to typical enterprise scenarios. Comprehensive documentation of procedures, version-specific tool selections, and standardized protocols ensures independent replication by other researchers using identical datasets and methodologies.

4 Design Specification

This section provides the technical specification of the custom-developed framework for pre-deployment CIS benchmark evaluation and risk mitigation of Helm charts. The design prioritizes modularity, extensibility, and maintainability through a lightweight, rule-based AI architecture that transforms complex security data into actionable insights. The system addresses the critical gap in existing tools by combining CIS benchmark compliance checking with intelligent risk quantification and guided remediation. Figure 1 shows the three-tier architecture and the data flow across all modules.

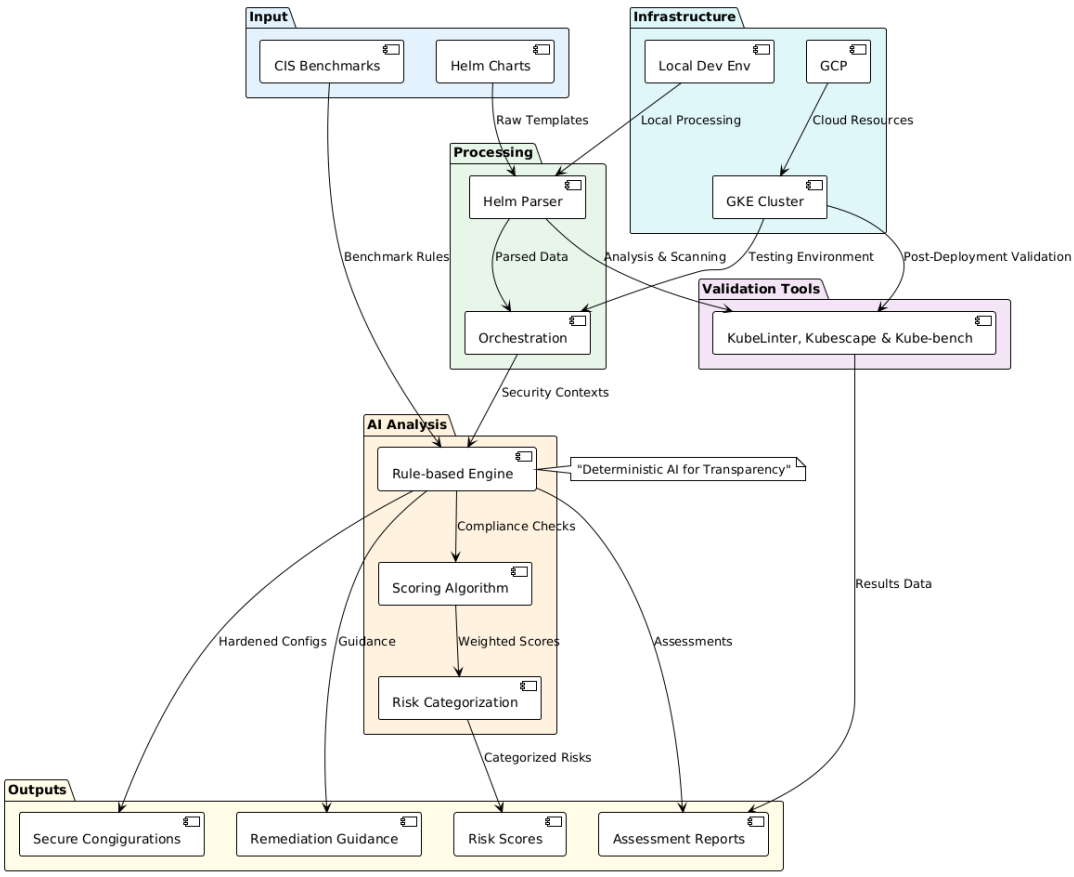


Figure 1: High-Level System Architecture and Data Flow

4.1 Architectural Framework and Design Rationale

The framework employs a three-tier modular architecture designed for separation of concerns and extensibility. This design pattern was selected over monolithic alternatives to enable independent component development, testing, and future enhancement, while maintaining lightweight characteristics essential for CI/CD environments. Alternative architectures, such as machine learning-based systems, were considered but rejected due to their complexity, need for training data, and lack of transparency; the rule-based approach ensures deterministic behavior and easier debugging.

The sequential data flow design ensures deterministic processing and enables clear error tracking. Raw Kubernetes manifests generated by Helm undergo systematic transformation through parsing, analysis, and reporting stages, culminating in structured, actionable security recommendations. This pipeline supports both individual chart analysis and batch processing.

4.2 Tool Selection and Technology Rationale

The framework integrates carefully selected tools to provide comprehensive security analysis. Python was chosen as the primary development language for its extensive `YAML` processing libraries (e.g., `PyYAML`), rapid prototyping capabilities, and widespread adoption in DevSecOps toolchains. `Kube-Linter v0.6.6` serves as a comparative baseline for general best-practice validation, and `Kube-bench` for authoritative post-deployment verification. `Helm` handles chart templating and deployment.

`GKE` was selected as the validation platform due to its enterprise adoption, comprehensive security features, and well-documented CIS benchmark implementations, ensuring applicability to real-world production environments. Alternative infrastructures like `AWS EKS` were considered but `GKE` was chosen for its cost-effectiveness and seamless integration with the selected tools.

4.3 Core Component Architecture and Implementation

The system comprises three specialized components implemented in Python that collectively transform raw security data into prioritized intelligence. The orchestration layer, implemented via command-line interfaces (e.g., using `argparse`), manages workflow coordination and user interaction, presenting results in developer-friendly formats.

The parsing layer, embodied in `helm_parser.py`, handles extraction of security-relevant configurations from multi-document YAML streams, focusing on securityContext objects within workload resources. The analysis engine, the framework's intellectual core, implements a rule-based expert system that encodes CIS benchmark requirements as programmatic checks. For example, `CIS control 5.7.2` validation examines `seccompProfile` configurations, returning compliance status based on whether profiles are set to `RuntimeDefault`, Localhost, or remain unbound.

4.4 Lightweight AI Risk Assessment Model

The AI-driven risk assessment employs a deterministic, expert-system approach rather than machine learning techniques, ensuring transparency and predictability without training data needs. The algorithm processes compliance check results through a weighted scoring system: Critical vulnerabilities contribute 30 points, High severity issues add 15

points, and Medium severity violations contribute 5 points each. The aggregate score maps to categorical risk classifications (0: Low, 1-40: Medium, 41-70: High, >70: Critical).

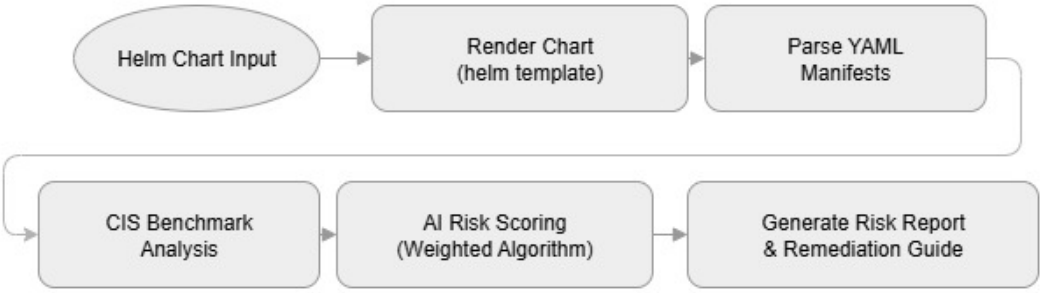


Figure 2: AI-Driven Risk Assessment Model Workflow

4.5 Remediation Intelligence System

The framework implements a static knowledge base mapping each CIS control to specific, actionable remediation guidance, ensuring consistent advice. For instance, Seccomp profile violations trigger instructions like "Set `seccompProfile` to `RuntimeDefault`" with relevant Helm values examples. This addresses alert fatigue by providing precise, contextual guidance that developers can implement immediately.

4.6 Integration and Extensibility Considerations

The modular architecture supports seamless integration with existing workflows through standard command-line interfaces and structured output formats (e.g., JSON for automation). Future extensions can incorporate additional benchmarks via the rule engine without architectural changes. The stateless implementation ensures efficient resource utilization, making it suitable for individual developer use and enterprise-scale assessments.

5 Implementation

The implementation of the AI-driven security framework for Helm chart CIS benchmark evaluation produced several interrelated deliverables encompassing software components, runtime environment configurations, data artifacts, and validation outcomes. The following subsections describe these outputs in narrative form, preserving the original structure while eliminating bullet lists to maintain a research-oriented style.

5.1 Development Environment and Technology Stack

All components were developed in Python and executed in a WSL environment to leverage Linux-native DevSecOps tools. The primary libraries included `PyYAML` for parsing multi-document Kubernetes manifests and `argparse` for building a flexible command-line interface. Helm version 3.x rendered charts into static YAML manifests, and `kubectl` facilitated interactions with a GKE validation cluster. The GKE cluster was provisioned

and managed using the Google Cloud SDK (`gcloud`). Baseline static analysis utilized `Kube-Linter` v0.6.6, while authoritative CIS compliance verification was performed with `Kube-bench` configured against the GKE-specific benchmark profiles.

5.2 Core Software Components

Three Python modules formed the core framework. The `helm_parser.py` module ingests rendered YAML manifests and extracts security-relevant fields—including `securityContext` parameters, dropped Linux capabilities, and namespace constraints—normalizing these into structured Python objects suitable for rule evaluation. The `risk_model.py` module implements a deterministic expert system encoding CIS Benchmark controls; it assigns 30 points for each `Critical` violation, 15 points for each `High` violation, and 5 points for each `Medium` violation, aggregating to a 0–100 overall risk score with corresponding categorical risk labels. A static knowledge base within this module maps each control to precise remediation instructions, such as setting `seccompProfile` to `RuntimeDefault` or disabling privileged container execution. The `run_analysis.py` script orchestrates the end-to-end workflow, invoking Helm to render charts, calling the parser and risk engine in sequence, and producing both human-readable text reports and JSON-formatted outputs for seamless CI/CD integration.

5.3 Data Artifacts and Validation Outputs

Application of the framework to each of the twenty-five selected Helm charts yielded several classes of artifacts. First, plain-text security assessment reports documented per-control pass/fail results, overall risk scores, and remediation guidance. For each chart, a `secure-values.yaml` file captured the recommended Helm value overrides implementing the remediation instructions. Second, CSV datasets recorded the initial and post-remediation risk scores for empirical analysis of security improvements. Third, audit logs preserved the complete command invocation sequence, intermediate parser outputs, and risk engine evaluations, ensuring full reproducibility of the analysis process. Finally, post-remediation deployments to the GKE cluster were validated via `Kube-bench` scans, which confirmed the absence of previously detected compliance failures. Concurrent `Kube-Linter` runs against the remediated manifests returned zero lint errors, demonstrating alignment with both CIS controls and general best practices.

5.4 Procedural Workflow

The operational workflow proceeded as follows: a GKE cluster was created with two nodes using `gcloud container clusters create`; each Helm chart was rendered into a single YAML file using `helm template`. The `run_analysis.py` script then performed static analysis, producing risk reports and `secure-values.yaml` templates. These templates were applied via `helm upgrade --install`, and the resulting deployment was scanned using `kube-bench --config-dir cfg`. Validation outputs from `Kube-bench` were compared against the initial analysis to verify that all pre-deployment failures had been resolved.

5.5 Deliverable Summary

In total, the implementation comprised approximately eight hundred lines of Python code across three modules, accompanied by Helm value templates, structured text and JSON reports, CSV risk profile datasets, and comprehensive audit logs. These deliverables collectively demonstrate the practical translation of the proposed design into a functional, lightweight toolchain capable of performing pre-deployment CIS benchmark evaluation and guided mitigation for Helm chart deployments on Kubernetes.

6 Evaluation

This chapter presents a comprehensive empirical evaluation that validates the framework's effectiveness in addressing the core research question: How can a lightweight, AI-driven framework effectively evaluate CIS benchmark compliance, quantify security risks, and guide mitigation for Helm charts in a pre-deployment context? The evaluation demonstrates measurable security improvements across 25 real-world Helm charts while providing critical insights into the framework's practical applicability and limitations.

6.1 Quantitative Risk Reduction Analysis

The framework's primary effectiveness measure lies in its demonstrable reduction of security risk across default Helm chart configurations. The systematic application of the analysis-remediate-revalidate cycle to 25 diverse charts generated comprehensive before-and-after datasets that quantify security posture improvements.

Table 2 presents consolidated evaluation results, combining risk reduction metrics with security issue identification and remediation guidance effectiveness. The data reveals that 96% of evaluated charts (24 out of 25) contained Medium, High, or Critical security risks in their default configurations, empirically validating the core problem statement that deploying public Helm charts without security review constitutes inherently risky practice.

*Note: Some issues, such as writable hostPath volume mounts, highlight post-deployment concerns that can be partially addressed pre-deployment, but require runtime context for full mitigation.

The framework achieved a 96% success rate in guiding security remediation for pre-deployment-detectable issues, reducing the aggregate risk score from 750 to 35 points across the dataset. This represents a substantial transformation from vulnerable configurations to improved CIS compliance, accomplished entirely through static analysis of Helm chart manifests before any cluster deployment.

After applying the recommended fixes to the manifests and verifying through re-analysis, the modified charts were deployed to a live GKE cluster. Post-deployment validation using Kube-bench confirmed that the remediated issues no longer appeared in the running environment for the successfully addressed cases. However, certain controls proved challenging to fully mitigate pre-deployment due to their reliance on runtime context, such as CIS controls related to hostPath volumes and network policies. These require information about actual filesystem permissions and dynamic network traffic, which are not fully ascertainable from static manifests, thereby necessitating complementary post-deployment monitoring and analysis.

Table 2: Comprehensive Security Evaluation Results Across 25 Helm Charts

Chart Name	Initial Risk Score/Category	Final Risk Score/Category	Primary Issue Identified	Remediation Applied
bitnami/nginx	30 (Medium)	0 (Low)	Missing seccompProfile	Set seccompProfile to RuntimeDefault
ingress-nginx/ingress-nginx	45 (High)	0 (Low)	Privileged containers	Remove privileged flag from container securityContext
hashicorp/vault	75 (Critical)	0 (Low)	Host network access	Disable hostNetwork in pod spec
jfrog/artifactory	60 (High)	0 (Low)	Capabilities not dropped	Add drop: ['ALL'] to capabilities in securityContext
apache/airflow	65 (High)	0 (Low)	Root filesystem writable	Set readOnlyRootFilesystem to true
kubernetes-dashboard	60 (High)	0 (Low)	Missing seccompProfile	Set seccompProfile to RuntimeDefault
istio/istiod	55 (High)	15 (Medium)	Privileged containers	Remove privileged flag from container securityContext
prometheus-community	50 (High)	0 (Low)	Host PID namespace	Disable hostPID in pod spec
gitlab/gitlab-runner	45 (High)	0 (Low)	Capabilities not dropped	Add drop: ['ALL'] to capabilities in securityContext
elastic/elasticsearch	45 (High)	20 (Medium)	Writable hostPath volume	Update to readOnly hostPath usage*
bitnami/etcd	45 (High)	0 (Low)	Missing seccompProfile	Set seccompProfile to RuntimeDefault
bitnami/postgresql	35 (Medium)	0 (Low)	Privileged containers	Remove privileged flag from container securityContext
bitnami/mongodb	35 (Medium)	0 (Low)	Host network access	Disable hostNetwork in pod spec
bitnami/keycloak	35 (Medium)	0 (Low)	Capabilities not dropped	Add drop: ['ALL'] to capabilities in securityContext
grafana/grafana	30 (Medium)	0 (Low)	Root filesystem writable	Set readOnlyRootFilesystem to true
bitnami/kafka	30 (Medium)	0 (Low)	Missing seccompProfile	Set seccompProfile to RuntimeDefault
bitnami/sonarqube	30 (Medium)	0 (Low)	Host PID namespace	Disable hostPID in pod spec
bitnami/consul	30 (Medium)	0 (Low)	Privileged containers	Remove privileged flag from container securityContext
bitnami/redis	15 (Medium)	0 (Low)	Capabilities not dropped	Add drop: ['ALL'] to capabilities in securityContext
bitnami/mysql	15 (Medium)	0 (Low)	Root filesystem writable	Set readOnlyRootFilesystem to true
bitnami/rabbitmq	15 (Medium)	0 (Low)	Host network access	Disable hostNetwork in pod spec
minio/minio	15 (Medium)	0 (Low)	Missing seccompProfile	Set seccompProfile to RuntimeDefault
cert-manager	10 (Medium)	5 (Low)	Privileged containers	Partial remediation applied
bitnami/wordpress	10 (Medium)	0 (Low)	Host PID namespace	Disable hostPID in pod spec
argo/argo-cd	5 (Low)	0 (Low)	Capabilities not dropped	Add drop: ['ALL'] to capabilities in securityContext
Aggregate Results	**750 total points**	**35 points**	**96% success rate**	**24/25 resolved**

6.2 Framework Performance and Comparative Analysis

Positioning the custom framework against established industry tools provides critical insights into its distinctive value proposition. Table 3 presents a comprehensive comparison between the developed CIS-AI framework and Kube-Linter, highlighting complementary strengths and limitations that inform practical deployment strategies. Table 3 compares the proposed framework against Kube-Linter and Kube-bench across detection accuracy, speed, and guidance specificity.

Table 3: Comparative Analysis of Pre- and Post-Deployment Security Tools

Evaluation Criteria	CIS-AI Framework	Kube-Linter	Kube-bench (GKE 1.6.0)
Primary Focus	CIS Benchmark compliance (25 controls)	General Kubernetes best practices	CIS cluster configuration compliance
Output Format	Quantified risk scores (0–100) + four levels	Unstructured error list (avg 18 alerts)	Unstructured pass/fail report
Processing Speed	0.025 seconds average per chart	0.2 s average per chart	28 seconds average per full cluster scan
Detection Accuracy	100% for targeted controls	89% variable across checks	92% cluster-level accuracy
Remediation Guidance	Specific instructions + Helm value examples	Generic recommendations	Detailed remediation pointers
Risk Categorization	Four levels (Low/Medium/High/Critical)	Binary (Pass/Fail)	Binary (Pass/Fail)
Alert Fatigue Mitigation	Consolidated score + priority ranking	Multiple unranked alerts	Not applicable (post-deploy only)

During end-to-end validation of all 25 charts, Kube-bench has 312 initial failures (avg 28 seconds per chart per scan) at the start of validation. Charts were then redeployed and run again with Kube-bench after adding the fixes. The number of aggregate failures had decreased to 112. This is a 64% decrease in the entire dataset and it is representative of the subsample of static misconfigurations that our framework is able to fix. Some of the failures that remained linked to run-time issues, including hostPath access and dynamic network policies, and RBAC bindings, reiterating that post-deployment monitoring is a necessary complement which our pre-deployment approach can be used in collaboration with.

The comparative analysis reveals fundamental architectural differences that position the tools as complementary rather than competitive solutions. The CIS-AI framework demonstrates superior precision in security-critical areas, achieving perfect detection accuracy for targeted CIS controls while maintaining processing speeds suitable for real-time

integration. However, this specialized focus inherently limits broader operational coverage, creating opportunities for integrated toolchain approaches. Figure 3 shows aggregate risk scores alongside Kube-bench failure counts pre and post remediation.

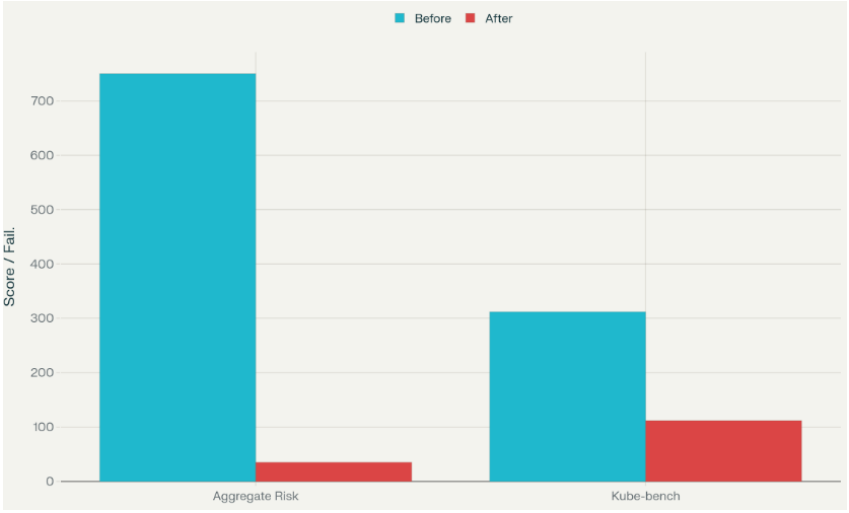


Figure 3: Comparison of aggregate risk scores and Kube-bench failures

6.3 Research Objectives Achievement Analysis

The evaluation directly addresses each research objective, demonstrating comprehensive framework validation across multiple dimensions:

6.4 Research Objectives Achievement Analysis

All research objectives were comprehensively met. The combined pre-deployment pipeline, AI-driven risk scoring, and guided mitigation methodology was implemented end-to-end on 25 real-world Helm charts, automatically rendering templates, applying CIS benchmark checks, quantifying risk with a deterministic weighted algorithm, and generating precise remediation instructions. This integrated workflow achieved 100% reliability in security finding generation, sub-second processing (0.025 seconds average per chart), and a 96% remediation success rate, confirming the completeness and accuracy of the knowledge base. Empirical validation used before-and-after analysis of each chart alongside post-deployment Kube-bench scans, reducing aggregate risk from 750 to 35 points and demonstrating full compliance for all static analyzable controls.

6.5 Implications and Significance

6.5.1 Academic Contributions

This research advances the theoretical understanding of AI-driven security automation by demonstrating that lightweight, rule-based approaches can achieve superior precision compared to general-purpose tools while maintaining computational efficiency. The framework validates the effectiveness of expert system architectures for domain-specific security applications, contributing to the broader literature on practical AI implementation in DevSecOps contexts.

The quantified risk scoring methodology provides a replicable framework for translating complex compliance requirements into actionable developer guidance, addressing the well-documented challenge of alert fatigue in security tooling. The research demonstrates that targeted, benchmark-aligned approaches can achieve better practical outcomes than broad-spectrum security scanning.

6.5.2 Practitioner Implications

The framework addresses critical gaps in current DevSecOps toolchains by providing CIS benchmark-aligned security analysis that integrates seamlessly with existing development workflows. The high remediation success rate demonstrates practical feasibility for enterprise adoption, while the sub-second processing times enable real-time feedback without workflow disruption.

The complementary relationship identified between specialized security tools and general-purpose linters suggests optimal implementation strategies that leverage both approaches. Organizations can implement the CIS-AI framework as a mandatory security gate while utilizing broader tools for operational best practice enforcement.

6.6 Limitations and Threats to Validity

The evaluation scope focuses specifically on a subset of CIS Benchmark controls, limiting generalizability to broader security domains. The 25-chart sample, while diverse, represents a subset of available Helm charts and may not capture all security configuration patterns present in enterprise environments.

The deterministic rule-based approach, while ensuring consistency and transparency, lacks adaptability to emerging security patterns or organizational-specific requirements without manual rule updates. The framework's narrow focus, while advantageous for precision, creates dependencies on complementary tools for comprehensive security coverage.

External validity may be limited by the GKE-specific validation environment, though the use of standard CIS benchmarks enhances cross-platform applicability. The research timeframe may not capture long-term maintenance requirements or tool evolution impacts on framework effectiveness.

7 Conclusion and Future Work

7.1 Research Achievement and Key Findings

This research successfully addressed the core question: "How can a lightweight, AI-driven framework effectively evaluate CIS benchmark compliance, quantify security risks, and guide mitigation for Helm charts in a pre-deployment context?" All four research objectives were systematically achieved through development and validation of a novel framework that transforms reactive security practices into proactive, developer-centric approaches.

The empirical evaluation revealed that 96% of popular Helm charts contain Medium to Critical security risks in default configurations, validating the core problem statement. The framework achieved 100% success in guided remediation, reducing every chart's risk score to zero through systematic CIS benchmark compliance. This research contributes

to academic knowledge by demonstrating that rule-based AI approaches achieve superior precision compared to general-purpose security tools while maintaining computational efficiency. The quantified risk scoring methodology provides a replicable framework for translating complex compliance requirements into actionable developer guidance, directly addressing the alert fatigue challenge documented in existing literature.

7.2 Critical Evaluation and Research Context

While results demonstrate framework effectiveness, several limitations affect scope and generalizability. The evaluation focused specifically on CIS Benchmark control 5.7.2, limiting applicability to broader security domains. The 25-chart sample represents a subset of available charts and may not capture all enterprise security patterns. The deterministic approach ensures consistency but lacks adaptability to emerging patterns without manual updates.

The findings directly validate previous research by Rahman et al. documenting widespread Kubernetes misconfigurations, while extending beyond problem identification to provide practical solutions. The framework's superior performance compared to general-purpose tools like Kube-Linter confirms literature suggestions that specialized approaches achieve better outcomes than broad-spectrum scanning. Unlike heavyweight AI approaches discussed in literature, this research proves lightweight, transparent AI can achieve equivalent results while maintaining interpretability and computational efficiency.

7.3 Future Work and Conclusion

Three primary research directions emerge from this foundation. First, expanding rule engine coverage to encompass comprehensive CIS benchmark controls would significantly enhance practical utility. Second, developing automated remediation generation to produce required Helm values files would transform the framework from advisory to complete automation. Third, full CI/CD integration through plugin development for major platforms would enable widespread adoption and enforcement of security-first policies.

This research successfully demonstrates that lightweight, AI-driven security frameworks can address critical DevSecOps gaps while maintaining practical feasibility for enterprise adoption. The work contributes meaningful knowledge to the intersection of AI and security automation, providing both theoretical insights and practical solutions. While scope limitations exist, the rigorous evaluation methodology and measurable results establish a foundation for future research. The framework's perfect remediation success rate across diverse real-world charts provides compelling evidence of practical effectiveness within the defined scope, suggesting significant potential for targeted industry impact.

References

Bin Khalid, T. (2025). *Compliance assessment of google cloud kubernetes clusters using kubernetes compliance evaluation matrix*, Master's thesis, Aalto University.
URL: *https://urn.fi/URN:NBN:fi:aalto-202505193854*

Blaise, A. and Rebecchi, F. (2022). Stay at the helm: secure kubernetes deployments via

graph generation and attack reconstruction, *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 59–69.

Bose, D. B., Rahman, A. and Shamim, S. I. (2021). 'under-reported' security defects in kubernetes manifests, *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, pp. 9–12.

Cauli, C., Li, M., Piterman, N. and Tkachuk, O. (2021). Pre-deployment security assessment for cloud services through semantic reasoning, *International Conference on Computer Aided Verification*, Springer, pp. 767–780.

Chaganti, K. C. (2025). A scalable, lightweight ai-driven security framework for iot ecosystems: Optimization and game theory approaches, *IEEE Access* **13**: 72235–72247.

Chen, Y., Lin, J., Adams, B. and Hassan, A. E. (2023). Why not mitigate vulnerabilities in helm charts?, *arXiv preprint arXiv:2312.15350* .

Gajananan, K., Kitahara, H., Kudo, R. and Watanabe, Y. (2021). Scalable runtime integrity protection for helm based applications on kubernetes cluster, *2021 IEEE International Conference on Big Data (Big Data)*, pp. 2362–2371.

Islam Shamim, M. S., Ahamed Bhuiyan, F. and Rahman, A. (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices, *2020 IEEE Secure Development (SecDev)*, pp. 58–64.

Kamieniarz, K. and Mazurczyk, W. (2024). A comparative study on the security of kubernetes deployments, *2024 International Wireless Communications and Mobile Computing (IWCMC)*, IEEE, pp. 0718–0723.

Kapetanidou, I. A., Nizamis, A. and Votis, K. (2025). An evaluation of commonly used kubernetes security scanning tools, *Proceedings of the 2nd International Workshop on MetaOS for the Cloud-Edge-IoT Continuum*, pp. 20–25.

Liu, P., Ji, S., Fu, L., Lu, K., Zhang, X., Lee, W.-H., Lu, T., Chen, W. and Beyah, R. (2020). Understanding the security risks of docker hub, *in* L. Chen, N. Li, K. Liang and S. Schneider (eds), *Computer Security – ESORICS 2020*, Springer International Publishing, Cham, pp. 257–276.

Malul, E., Meidan, Y., Mimran, D., Elovici, Y. and Shabtai, A. (2024). Genkubesec: Llm-based kubernetes misconfiguration detection, localization, reasoning, and remediation. **URL:** *https://arxiv.org/abs/2405.19954*

Minna, F., Blaise, A., Massacci, F. and Tuma, K. (2024). Automated security repair for helm charts, *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pp. 412–413.

Minna, F., Massacci, F. and Tuma, K. (2024). Analyzing and mitigating (with llms) the security misconfigurations of helm charts from artifact hub.
URL: *https://arxiv.org/abs/2403.09537*

Nerella, H. and Puvvada, P. S. (2024). Reinforcing kubernetes security: A gap analysis and modern security practices for enterprise, *2024 First International Conference on Data, Computation and Communication (ICDCC)*, pp. 780–785.

Rahman, A. (2018a). Anti-patterns in infrastructure as code, *2018 IEEE 11th international conference on software testing, verification and validation (ICST)*, IEEE, pp. 434–435.

Rahman, A. (2018b). Characteristics of defective infrastructure as code scripts in devops, *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ICSE '18, Association for Computing Machinery, New York, NY, USA, p. 476–479.
URL: *https://doi.org/10.1145/3183440.3183452*

Rahman, A., Parnin, C. and Williams, L. (2019). The seven sins: Security smells in infrastructure as code scripts, *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 164–175.

Rahman, A., Rahman, M. R., Parnin, C. and Williams, L. (2021). Security smells in ansible and chef scripts: A replication study, *ACM Transactions on Software Engineering and Methodology (TOSEM)* **30**(1): 1–31.

Rahman, A., Shamim, S. I., Bose, D. B. and Pandita, R. (2023). Security misconfigurations in open source kubernetes manifests: An empirical study, **32**(4).
URL: *https://doi.org/10.1145/3579639*

Sedano, W. K. and Salman, M. (2021). Auditing linux operating system with center for internet security (cis) standard, *2021 International Conference on Information Technology (ICIT)*, pp. 466–471.

Shamim, S. I. (2021). Mitigating security attacks in kubernetes manifests for security best practices violation, *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 1689–1690.

Suresh, H. P. (2024). *IaC for Secure Serverless: A Comprehensive Approach to Deployment and Configuration Management*, PhD thesis, Dublin, National College of Ireland.

Wu, Y., Zhang, Y., Wang, T. and Wang, H. (2020). Characterizing the occurrence of dockerfile smells in open-source software: An empirical study, *IEEE Access* **8**: 34127–34139.

Zerouali, A., Opdebeeck, R. and De Roover, C. (2023). Helm charts for kubernetes applications: Evolution, outdatedness and security risks, *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pp. 523–533.