Journal of Computing and Security

Research Article, 2026, Volume 13, Number 1 (pp. 73–82)

http://www.jcomsec.org

A Comparative Study of AI-Generated and Human-Written Kubernetes Manifests

Nasim Tabibpour [a] Mohammad Dakhilalian [a,*]

[a] *Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan 84156-83111, Iran.*

ARTICLE INFO.

Article history:
Received: 05 January 2026
Revised: 08 May 2026
Accepted: 04 July 2026
Published Online: 29 July 2026

Keywords:
Cloud-Native Security; Configuration Smells; Container Orchestration; Large Language Models; Static Analysis.

ABSTRACT

Kubernetes is susceptible to security and quality issues when misconfigured; meanwhile, the use of generative models such as ChatGPT to produce manifests is rising. This study compares the configuration quality of human-authored manifests (1,985 open-source files) with a published 2023 corpus of AI-generated manifests (98 ChatGPT-generated files aligned with a prior MSR study). Using an aligned KubeLinter-based framework, we analyze six configuration smells and report density (occurrences per 1,000 raw lines) and proportion (percentage of files affected). We also add a balanced-size robustness check that subsamples 98 human-written manifests 1,000 times. The results show that missing CPU and memory specifications remain more prevalent in the AI corpus under this balanced comparison. In contrast, omissions in secure execution, such as missing `runAsNonRoot` and `readOnlyRootFilesystem`, are important in both corpora. An issue-level smell-by-kind analysis of the human corpus shows that 85.16% of target-smell occurrences concentrate in `Deployment` and `StatefulSet` resources. The findings provide a focused empirical baseline and support the use of human review, static analysis, and policy enforcement before integration or deployment.

1 Introduction

Kubernetes is an open-source platform for the automated management of compute services such as containers [1]. Containers help organizations manage application lifecycles efficiently [1]. In large software projects, thousands of containers may run concurrently, each requiring monitoring, updates, resource configuration, and secure communication with other services. Doing this manually is time-consuming and error-prone. "Container orchestration" refers to the intelligent management of these lifecycles using platforms like Kubernetes [2]. For example, Capital One reported a substantial increase in release rates after adopting Kubernetes [3]. Kubernetes has become the most popular orchestrator, with a projected market size of $7.8B by 2030 [4].

However, as a complex and highly configurable system, Kubernetes is sensitive to security misconfigurations [5]. Given Kubernetes' broad adoption, misconfigurations can entail serious consequences. The State of Kubernetes Security Report (2021) notes that 94% of 500 surveyed professionals experienced at least one Kubernetes-related security incident, most due to misconfigurations [6]. Consequently, auditing and fixing misconfigurations in Kubernetes manifests—the

* Corresponding author.
Email addresses: `n.tabibpour@alumni.iut.ac.ir` (N. Tabibpour), `mdalian@iut.ac.ir` (M. Dakhilalian)
https://dx.doi.org/10.22108/jcs.2026.148046.1195
ISSN: 2322-4460

YAML/JSON files that declare resource behavior—are crucial [1, 7]. Studying real-world manifests can reveal the frequency and nature of issues [5, 8], because many practitioners lack the security expertise to identify and remediate them [6, 9].

In parallel, generative AI (e.g., ChatGPT) has grown rapidly across software-engineering tasks. These models can produce or complete Kubernetes manifests from natural-language prompts, potentially accelerating development [10–12]. Still, LLM-generated outputs may contain quality or security issues [13]. Zhang et al. [2] evaluated ChatGPT-generated Kubernetes manifests using static analyzers, including KubeLinter [14], and reported notable misconfigurations.

This observation raises the question of how the documented issues in AI-generated outputs relate to the long-standing misconfigurations found in human-written manifests. Building on that line, this study compares a large, re-collected human-written corpus with the published KubeLinter-aligned measurements of the AI-generated corpus studied by Zhang et al. [2]. The revised research questions are:

- **RQ1:** To what extent do AI-generated manifests differ from human-written ones in the prevalence and density of aligned security and configuration smells?
- **RQ2:** Which misconfigurations are most prevalent in each group?
- **RQ3:** Where do target configuration smells concentrate across Kubernetes resource kinds in the human-written corpus?

The third question is intentionally scoped to the human-written corpus, where our reprocessing provides exact issue-level smell-by-kind mappings. The AI corpus is used through published aggregate measurements, so we avoid unsupported AI resource-kind claims.

This paper makes three key contributions. First, it provides a systematic comparison between human-written Kubernetes manifests and a published 2023 ChatGPT-generated baseline under an aligned KubeLinter-based framework. Second, it adds a balanced-size robustness check using repeated 98-file human subsamples. Third, it provides an issue-level smell-by-kind analysis for the human-written corpus, showing that target smells concentrate primarily in workload resources such as `Deployment` and `StatefulSet`. These results support actionable recommendations for static analysis, CI/CD quality gates, and Kubernetes policy enforcement.

The remainder of this paper is organized as follows. Section 2 reviews background and related work. Section 3 describes the datasets, analysis steps, and measurement framework. Section 4 reports the findings that answer RQ1–RQ3. Sections 5 and 6 discuss implications, limitations, and conclusions.

2 Background and Related Works

Prior studies have investigated the security and quality issues that arise from Kubernetes misconfigurations. Rahman et al. [5] conducted one of the most extensive empirical studies by analyzing over 2,000 manifests from 92 GitHub and GitLab repositories using SLI-KUBE, a static analyzer tailored for Kubernetes. Their work showed that misconfigurations related to privilege escalation, missing resource constraints, and insecure defaults were widespread in real-world deployments. They further reported that these issues often propagate through CI/CD pipelines, making them difficult to detect without automated tooling. However, their focus was on human-authored manifests and did not examine whether newer development practices—such as the use of generative AI—introduce different patterns of misconfiguration.

Bose et al. [8] performed a comprehensive study of Kubernetes security vulnerabilities, highlighting under-reported defects in configuration and access control. Their analysis revealed that many issues stem not from Kubernetes itself but from incorrectly specified manifests, including weak security contexts and improper network exposure. While their work provided valuable insights into the types of configuration vulnerabilities that occur in production clusters, it did not compare different authorship sources using a unified measurement framework.

Shamim et al. [15] synthesized best practices for secure Kubernetes deployment through a systematic review of grey literature. They formulated the "XI Commandments," a set of prescriptive guidelines such as enforcing least-privilege execution, specifying resource limits, and avoiding unnecessary service exposure. Although these guidelines are influential and widely cited, the authors did not empirically evaluate how often such recommendations are violated across AI-generated and human-written manifests.

With the increasing use of generative AI, Zhang et al. [2] studied ChatGPT-generated Kubernetes manifests and reported frequent omissions of CPU and memory resource specifications, weak security defaults, and errors in pod–service wiring. Using KubeLinter as one of the static analyzers, they showed that LLMs can accelerate configuration writing but often produce insecure or incomplete manifests. Their study provides the AI-side baseline that motivates our aligned comparison with a larger human-written corpus.

Across these studies, a focused comparison remains useful. Prior work has studied either human-written Kubernetes manifests or AI-generated manifests, but less is known about how the two sources compare when evaluated through the same smell taxonomy and normalized metrics. To address this gap, our work compares the published AI-manifest baseline from Zhang et al. [2] with a re-collected human-written corpus following Rahman et al. [5]. We use the 2023 AI corpus as a historical empirical baseline rather than as evidence about all current LLMs.

3 Methodology

This section details the study pipeline: (i) data selection and preparation, (ii) validation and cleaning, (iii) static analysis of manifests to extract security and quality smells, (iv) metric definitions, and (v) the balanced-size robustness check used to address dataset imbalance.

3.1 Study Design and Comparison Units

We compare two datasets to answer the research questions:

- **AI group:** ChatGPT-generated manifests aligned with the DevGPT data used in the MSR'24 study of Zhang et al. [2]. That work examined six snapshots from July–August 2023, filtered the collected snippets to 98 valid Kubernetes manifests, and reported KubeLinter-based results using a security/network taxonomy. We adopt the same smell definitions and use the published AI-side measurements as the aligned reference baseline.
- **Human group:** Open-source human-written manifests collected following Rahman et al. [5]. Their reference set contains 2,039 manifests across 92 repositories. In our re-collection, 54 manifests (2.65%) could not be retrieved because some original links, files, or repositories were no longer publicly accessible. The final human-written set therefore contains **1,985** files [16].

Dataset imbalance and scope. The two corpora are asymmetric in size: the human-written set (1,985 files) is an order of magnitude larger than the AI-generated set (98 files). This reflects the availability and scale of prior datasets rather than a design choice to balance sample sizes. A larger human corpus improves coverage of real-world usage, but it also means that raw counts are not directly comparable across groups. We therefore report normalized metrics—*density* and *proportion*, defined in Section 3.5—and complement them with the balanced-size robustness check in Section 3.6. These metrics reduce, but do not eliminate, concerns related to dataset imbalance and file-length differences.

Scope of the AI corpus. The AI-generated corpus reflects ChatGPT outputs collected in July–August 2023. Because LLM behavior evolves rapidly, the results should not be interpreted as a claim about all current generative models. Instead, the AI corpus is treated as a historical baseline for early ChatGPT-generated Kubernetes manifests. Re-running the generation process with newer models and documented prompt strategies is left to future work.

Unavailable human files. The 54 unavailable human manifests were excluded only because they could not be retrieved from the original public locations. We did not apply content-based exclusions based on project type, resource kind, smell presence, or cloud provider. Nevertheless, because inaccessible files cannot be fully inspected after link rot or repository removal, we cannot rule out systematic differences between accessible and inaccessible files. We therefore treat the re-collected human corpus as a large open-source empirical sample rather than a perfectly representative benchmark.

3.2 Rationale for Smell Selection

The six configuration smells analyzed in this study were selected to preserve comparability with the prior AI-generated manifest study by Zhang et al. [2]. Since the central objective is to compare human-written manifests against the AI-generated baseline under an aligned framework, we restrict the main analysis to smell categories that can be consistently measured using the same static analyzer and rule definitions. This choice improves internal validity and avoids comparing results produced by different taxonomies.

We acknowledge that Kubernetes misconfiguration is broader than these six smells. Important additional risks include privilege escalation, misuse of `hostPath`, Linux capability misconfiguration, insecure service exposure through `NodePort` or `LoadBalancer`, and exposure of secrets or sensitive data. These issues are outside the aligned comparison scope of the present study and are therefore treated as future work rather than as evidence that they are unimportant.

3.3 Target Smells and Static Analysis

We analyze six aligned misconfigurations using KubeLinter [14], matching the MSR'24 categories:

- **Network:** Dangling Service, Absent Anti-affinity.
- **Security/reliability:** Absent Read-only

Filesystem, Run as Non-root (disabled), Unset CPU Requirements, Unset Memory Requirements.

KubeLinter was used because it is the analyzer used in the prior AI-generated manifest study and therefore provides a common measurement instrument for comparing the AI and human corpora. Using additional tools could broaden coverage, but it would also introduce different rule definitions, severity models, and detection heuristics. We therefore use KubeLinter as the aligned analyzer for the main comparison and discuss single-tool dependence as a threat to validity.

Prior work reports that manifest misconfigurations can adversely impact the robustness, security, and availability of cloud-native systems [7]. Below we operationalize each of the six smells to make our monitoring and comparison criteria explicit.

Dangling Service. A Service is declared in the manifest, but no Pod backs it (e.g., `spec.selector` does not match any pod labels). This yields useless endpoints, unnecessary network resource consumption, and extra operational complexity that hampers observability and troubleshooting [7].

Absent Anti-affinity. No anti-affinity policy (`podAntiAffinity`) is specified, so pods may colocate on the same node or zone, reducing failure tolerance and potentially degrading performance due to resource contention. A single node failure can then disrupt multiple replicas simultaneously [7].

Absent Read-only Filesystem. The root filesystem is writable because `readOnlyRootFilesystem=true` in `securityContext` is unset. This deviates from least-privilege hardening and increases the risk of unauthorized or persistent changes to the container image [7].

Run as Non-root (disabled). Processes run with root privileges when `runAsNonRoot=true` in `securityContext` (and ideally a nonzero `runAsUser`) is not enforced. Running as root amplifies the blast radius of exploits or vulnerable binaries; restricting privileges is a widely recommended best practice [7].

Unset CPU Requirements. Missing CPU requests or limits impairs scheduling and capacity planning, increasing the risk of overcommit and unpredictable performance under load [7].

Unset Memory Requirements. Missing memory requests or limits lead to poor memory management and a higher likelihood of excessive memory consumption, OOMKill events, or pod crashes, undermining service reliability [7].

3.4 Processing Human-written Manifests

We implemented a Python script that, for each YAML file, runs KubeLinter and parses its output line by line. If any of the six target smells appear, we record the file identifier, smell type, raw KubeLinter line, affected resource kind, and raw line count. This issue-level extraction enables both per-file metric computation and smell-by-kind analysis. We then aggregate the results into structured tables for reporting.

3.5 Metrics and Definitions

To make results comparable across corpora with different sizes and heterogeneous file lengths, we use two complementary metrics for each smell x: (i) *density*, which captures how often a smell occurs per unit of manifest size, and (ii) *proportion*, which captures how widely a smell is spread across files. Formally, for each smell x and each group, we compute density and proportion as follows:

$$\text{Density}(x) = \frac{\text{total occurrences of } x}{\text{total raw lines across manifests}} \times 1000 \tag{1}$$

$$\text{Proportion}(x) = \frac{\#\text{ manifests with at least one occurrence of } x}{\text{total } \#\text{ manifests}} \times 100. \tag{2}$$

Density, as shown in Equation (1), is normalized by the total number of raw lines and therefore reflects smell intensity per 1,000 lines of manifest code. Proportion (Equation (2)) measures breadth by indicating what fraction of files are affected at least once. However, density can still be influenced by file length and manifest complexity, so we interpret it together with proportion and the balanced-size robustness check rather than treating density alone as conclusive evidence.

3.6 Balanced-Size Robustness Check

Because the human-written corpus is substantially larger than the AI-generated corpus, we perform a balanced-size robustness check. We randomly sample 98 human-written manifests without replacement and repeat this procedure 1,000 times. For each subsample, we recompute the proportion and density of each target smell. We then report the empirical mean and the 2.5th and 97.5th percentiles of the resulting distributions. This analysis does not replace the full-corpus results; rather, it tests whether the observed patterns remain visible when the human corpus is evaluated at the same file count as the AI corpus.

3.7 Mapping to Research Questions

RQ1: For each smell and group, compute proportion and density and compare side by side, supported by

the balanced-size robustness check.
RQ2: Rank smells within each group by proportion and density to identify dominant misconfigurations.
RQ3: Map each target-smell occurrence in the human-written corpus to its Kubernetes resource kind and report the issue-level smell-by-kind distribution.

4 Results

We build on the data and methodology described in Section 3 and proceed through RQ1–RQ3. We first compare the aligned AI and human metrics, then report the balanced-size robustness check, identify dominant smells, and analyze the issue-level distribution of human-written smells across Kubernetes resource kinds.

4.1 Answer to RQ1: Overall Comparison

Table 1 shows the per-smell metrics for the aligned AI and human corpora. In the AI-generated corpus, the most prominent issues are *Unset CPU Requirements* and *Unset Memory Requirements*: both appear in 35.8% of files and have a density of 45.5 occurrences per 1,000 lines. In the human-written corpus, these two resource-requirement smells are also present, but their full-corpus proportions are lower: 22.92% for CPU and 24.08% for memory.

Secure-execution smells are important in both corpora. In the human-written corpus, *Absent Read-only Filesystem* and *Run as Non-root* are the two highest-proportion smells, affecting 29.47% and 28.72% of files, respectively. The corresponding AI proportions are 35.8% and 35.7%. Therefore, these smells should not be interpreted as exclusive to one authorship source; instead, they represent recurring hardening omissions across both human-written and AI-generated manifests.

Network-related smells are overall less common. *Dangling Service* appears in 11.69% of human manifests and 10.5% of AI manifests, while *Absent Anti-affinity* is the rarest smell in both datasets. These results suggest that basic resource and execution defaults are more frequently omitted than placement-policy hardening rules.

4.2 Balanced-Size Robustness Check

Table 2 reports the balanced-size robustness check. The key resource-specification finding remains robust. For *Unset CPU Requirements*, the AI proportion is 35.8%, while the 95% empirical interval from 1,000 balanced human subsamples is [15.31, 31.63]. For *Unset Memory Requirements*, the AI proportion is again 35.8%, while the corresponding human interval is [16.33, 32.65]. Thus, even when the human corpus is repeatedly evaluated at the same sample size as the AI corpus, missing CPU and memory specifications remain more prevalent in the AI baseline.

For secure-execution smells, the interpretation is more nuanced. The AI proportions for *Absent Read-only Filesystem* and *Run as Non-root* are close to or within the empirical ranges observed in balanced human subsamples. Therefore, we do not treat these two smells as a strong sample-size-robust difference in file-level prevalence. Instead, the results indicate that secure-execution omissions are important in both corpora. Density values are higher in the AI baseline for all six smells; however, density is normalized by raw lines and can be affected by file length and manifest complexity. We therefore interpret density together with proportion and balanced-size analysis.

4.3 Answer to RQ2: Dominant Smells per Group

The dominant smells differ depending on whether we rank by within-corpus prevalence or by the balanced AI–human comparison. In the AI-generated corpus, *Unset CPU Requirements* and *Unset Memory Requirements* are dominant by both proportion and density, and their AI proportions are above the empirical 95% ranges from equal-size human subsamples.

In the human-written corpus, *Absent Read-only Filesystem* and *Run as Non-root* have the highest full-corpus proportions. However, because the AI proportions for these smells lie within or close to the balanced human subsampling ranges, we interpret them as important cross-corpus hardening issues rather than as a strong authorship-specific difference. *Absent Anti-affinity* remains the rarest smell in both datasets.

4.4 Answer to RQ3: Issue-Level Resource-Kind Analysis in the Human Corpus

To answer the revised RQ3, we mapped each detected target-smell occurrence in the human-written corpus to the Kubernetes resource kind reported by KubeLinter. This issue-level mapping allows us to examine where misconfigurations occur in the Kubernetes object structure. The two tables below summarize the resource-kind distribution and the smell-by-kind cross-tabulation.

The results show that target smells are highly concentrated in workload-level resources. In the resource-kind distribution table, `Deployment` accounts for 2,033 occurrences, corresponding to 54.34% of all target-smell occurrences in the human-written corpus. `StatefulSet` is the second most affected kind, with 1,153 occurrences, corresponding to 30.82%. Together,

Table 1. Per-smell metrics for both corpora. Density denotes occurrences per 1,000 raw lines. Human totals: #files=1,985, raw lines=183,220. AI metrics are the aligned published baseline from Zhang et al. [2].

Smell		Human				AI		
Category	**Name**	**Prop.**	**Density**	**Count**	**#Files ≥1**	**Prop.**	**Density**	**Count**
Network	Dangling Service	11.69	1.32	241	232	10.50	6.70	10
Network	Absent Anti-affinity	3.83	0.48	88	76	2.70	4.20	4
Security	Absent Read-only Filesystem	29.47	5.06	928	585	35.80	22.70	34
Security	Run as Non-root (disabled)	28.72	4.91	899	570	35.70	22.80	34
Security	Unset CPU Requirements	22.92	4.25	779	455	35.80	45.50	68
Security	Unset Memory Requirements	24.08	4.40	806	478	35.80	45.50	68
	Total occurrences	–	20.42	3741	–	–	146.00	218

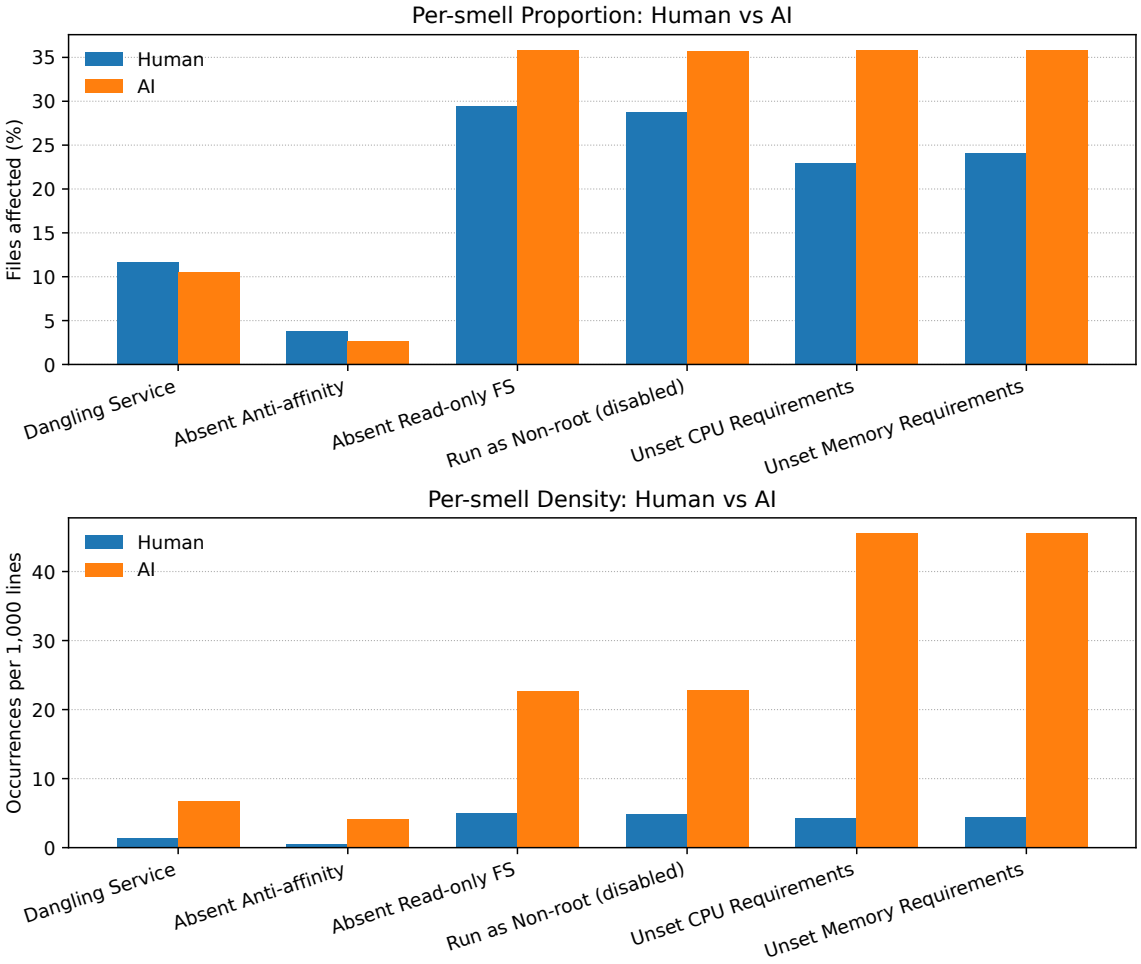

Figure 1. Per-smell differences between human and AI manifests. Top: proportion of files affected (%). Bottom: density (occurrences per 1,000 lines). See Table 1 for numeric values.

`Deployment` and `StatefulSet` account for 85.16% of all detected target smells.

Table 4 provides a more detailed breakdown. *Dangling Service* appears exclusively in `Service` resources, as expected from the semantics of the

Table 2. Balanced-size robustness check using 1,000 random subsamples of 98 human-written manifests. Intervals report empirical 2.5th and 97.5th percentiles.

Smell	Human Prop.	Human Subsample Prop.	AI Prop.	Human Density	Human Subsample Density	AI Density
Dangling Service	11.69	11.74 [6.10, 18.37]	10.50	1.32	1.71 [0.33, 3.29]	6.70
Absent Anti-affinity	3.83	3.82 [1.02, 8.16]	2.70	0.48	0.63 [0.04, 1.67]	4.20
Absent Read-only Filesystem	29.47	29.49 [21.43, 38.78]	35.80	5.06	6.46 [1.33, 11.35]	22.70
Run as Non-root	28.72	28.69 [20.41, 37.76]	35.70	4.91	6.28 [1.22, 11.21]	22.80
Unset CPU Requirements	22.92	22.86 [15.31, 31.63]	35.80	4.25	5.41 [1.06, 9.83]	45.50
Unset Memory Requirements	24.08	24.04 [16.33, 32.65]	35.80	4.40	5.61 [1.09, 9.92]	45.50

Table 3. Distribution of target-smell occurrences across resource kinds in the human-written corpus.

Kind	Count	Share (%)
Deployment	2033	54.34
StatefulSet	1153	30.82
Service	241	6.44
Pod	208	5.56
CronJob	40	1.07
Job	30	0.80
DaemonSet	18	0.48
ReplicationController	13	0.35
ReplicaSet	5	0.13
Total	3741	100.00

smell. In contrast, secure-execution and resource-requirement smells concentrate mainly in `Deployment` and `StatefulSet`. For example, *Absent Read-only Filesystem* appears 556 times in `Deployment` and 292 times in `StatefulSet`; *Run as Non-root* appears 533 times in `Deployment` and 289 times in `StatefulSet`. The same pattern is visible for *Unset CPU Requirements* and *Unset Memory Requirements*.

This concentration follows from the structure of Kubernetes workload resources. `Deployment` and `StatefulSet` objects often contain pod templates with one or more containers, so omissions in fields such as `resources.requests`, `resources.limits`, `runAsNonRoot`, and `readOnlyRootFilesystem` can repeat within the same manifest. These results suggest that continuous linting and policy enforcement should prioritize workload resources in the human-written corpus.

4.5 Summary and Actionable Message

The revised analysis leads to a more cautious interpretation than a simple raw-size comparison. Missing CPU and memory specifications are the most robust AI-side findings, while secure-execution omissions remain important in both corpora. Density values highlight intensity but must be interpreted cautiously because shorter manifests can produce high density even with few absolute occurrences. Overall, AI-assisted Kubernetes generation should be combined with human review, static linting, and policy enforcement before deployment.

5 Discussion

This section synthesizes the implications of the revised empirical findings and discusses how they relate to Kubernetes configuration quality, practical deployment risk, and threats to validity.

5.1 Interpretation of Findings

The results indicate that Kubernetes manifests remain prone to recurring security and reliability omissions. The balanced-size analysis strengthens the main resource-management finding: missing CPU and memory specifications are the most robust AI-side weakness in the aligned comparison. This suggests that AI-generated manifests may require explicit prompting or post-generation validation for resource requests and limits.

At the same time, the secure-execution findings require a more cautious interpretation. Missing `runAsNonRoot` and `readOnlyRootFilesystem` are common in the human-written corpus and appear frequently in the AI baseline. Rather than assigning these issues exclusively to one authorship source, the results suggest that secure-execution defaults are often omitted unless they are explicitly required by templates, policies, or review processes.

The resource-kind analysis further shows that human-written configuration smells concentrate in workload-level resources, especially `Deployment` and `StatefulSet`. This is consistent with Kubernetes object structure: these resources contain pod templates, container specifications, and security-context fields

Table 4. Cross-tabulation of target configuration smells by Kubernetes resource kind in the human-written corpus.

Smell	CronJob	DaemonSet	Deployment	Job	Pod	ReplicaSet	ReplicationController	Service	StatefulSet	Total
Dangling Service	0	0	0	0	0	0	0	241	0	241
Absent Anti-affinity	0	0	84	0	0	1	1	0	2	88
Absent Read-only Filesystem	10	6	556	8	52	1	3	0	292	928
Run as Non-root	10	5	533	6	52	1	3	0	289	899
Unset CPU Requirements	10	3	419	8	52	1	3	0	283	779
Unset Memory Requirements	10	4	441	8	52	1	3	0	287	806
Total	40	18	2033	30	208	5	13	241	1153	3741

where the analyzed smells are expressed.

5.2 Runtime and Security Implications

Although this study is based on static analysis, the detected smells have direct operational implications. Missing CPU requests and limits can lead to poor scheduling decisions, resource contention, and unstable performance under load. Missing memory requirements can increase the likelihood of excessive memory consumption, OOMKill events, and pod restarts. Missing `runAsNonRoot` weakens privilege isolation and increases the blast radius of a compromised container. Similarly, a writable root filesystem can allow unauthorized changes inside the container environment, which conflicts with least-privilege hardening practices.

Network-related smells also have practical consequences. A dangling `Service` may expose a stable service object without a valid backend, leading to unreachable applications and additional debugging overhead. Missing anti-affinity does not necessarily cause immediate failure, but it may reduce fault tolerance by allowing replicas to be co-located on the same node or failure domain. Thus, even when detected through static analysis, these smells correspond to realistic reliability and security risk conditions in deployed Kubernetes environments. We do not claim that every detected smell leads to an immediate runtime failure; rather, the smells indicate potential risk conditions that should be reviewed before deployment.

5.3 Practical Implications

From a practitioner perspective, the findings point to several actionable recommendations. AI-generated manifests should be checked for CPU and memory requests/limits, while both AI-generated and human-written manifests should be subject to secure-execution defaults such as `runAsNonRoot` and `readOnlyRootFilesystem`. These rules can be enforced through CI/CD checks, templates, policy-as-code, or admission-control mechanisms. For human-written manifests, workload resources such as `Deployment` and `StatefulSet` deserve particular attention because they account for most detected target-smell occurrences.

Combining Kubernetes domain expertise with generative AI and static analysis offers a practical route to safer cluster configurations. The results support using generative models as assistants rather than autonomous configuration authorities.

5.4 Threats to Validity

This study has several limitations. First, the two corpora differ substantially in size. The human-written corpus contains 1,985 manifests, whereas the AI baseline contains 98 manifests. We address this concern by avoiding raw-count comparisons and by adding a balanced-size robustness check with 1,000 random 98-file human subsamples. Nevertheless, the AI corpus remains small, and a single misclassified AI manifest can affect file-level proportions more than a single misclassified human manifest.

Second, the AI corpus reflects ChatGPT outputs collected in July–August 2023. LLM behavior changes rapidly, so the findings should not be generalized to all current models or prompting strategies. The revised manuscript therefore treats the AI corpus as a historical empirical baseline. Future work should repeat the generation process with newer models, controlled prompt strategies, and transparent generation parameters.

Third, the human corpus is a re-collected subset of the reference corpus used by Rahman et al. [5]. We retrieved 1,985 of the original 2,039 manifests; 54 files (2.65%) were unavailable because some original links, files, or repositories were no longer publicly accessible. We did not exclude files based on content, smell presence, or project category. However, because inaccessible repositories cannot be fully inspected after link rot or removal, we cannot rule out systematic bias in the missing portion. This limitation is now explicitly acknowledged.

Fourth, the analysis is limited to six configuration

smells. This was an intentional methodological decision to preserve comparability with the prior AI-manifest study, but Kubernetes misconfiguration is broader than these six rules. Future work should extend the taxonomy to include privilege escalation, `hostPath` misuse, Linux capabilities, insecure service exposure, and secrets exposure.

Fifth, the study relies on a single static analysis tool, KubeLinter. This may introduce tool-specific bias because different Kubernetes scanners implement different rules and detection heuristics. However, using KubeLinter was necessary to preserve comparability with the prior AI-generated manifest baseline. Future work should replicate the analysis with additional tools such as kube-score, Checkov, Kubescape, Datree, Trivy, or Polaris.

Finally, the study evaluates manifests through static analysis and does not execute them in a real or simulated Kubernetes cluster. Therefore, the results identify potential configuration risks rather than directly measured runtime failures or exploitability. A controlled deployment-based validation would be valuable for assessing the operational impact of the detected smells.

6 Conclusions

This paper presented a systematic comparison between human-written Kubernetes manifests and a published 2023 AI-generated manifest baseline under an aligned KubeLinter-based framework. The revised analysis shows that missing CPU and memory specifications are the most robust AI-side weakness when compared against balanced 98-file human subsamples. Secure-execution omissions, including missing `runAsNonRoot` and `readOnlyRootFilesystem`, are important in both corpora and should be treated as general Kubernetes hardening priorities. The issue-level resource-kind analysis further shows that most human-written target smells concentrate in `Deployment` and `StatefulSet` resources. Overall, the findings highlight the importance of continuous linting, policy enforcement, security-aware templates, and human oversight when integrating generative AI into Kubernetes configuration workflows.

Conflict of Interest

The authors declare that there is no conflict of interest.

References

[1] S. Miles. *Kubernetes: A Step-By-Step Guide For Beginners To Build, Manage, Develop, and Intelligently Deploy Applications (2020 Edition)*. Independently Published, 2020.

[2] Y. Zhang, R. Meredith, W. Reeves, J. Coriolano, M. A. Babar, and A. Rahman. Does generative AI generate smells related to container orchestration? an exploratory study with Kubernetes manifests. In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR)*, pages 192–196, 2024. doi:10.1145/3643991.3645079.

[3] Kubernetes. Case study: Capital one. https://kubernetes.io/case-studies/capital-one/, 2023. Accessed: 2023-06-10.

[4] Markets N Research. Latest global Kubernetes market size & share: USD 7.8 billion by 2030. https://www.globenewswire.com/news-release/2023/03/06/2621358/0/en/Latest-Global-Kubernetes-Market-Size-Share-Worth-U html, 2023. Accessed: 2023-06-12.

[5] A. Rahman, S. I. Shamim, D. B. Bose, and R. Pandita. Security misconfigurations in open source Kubernetes manifests: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 32(4):1–36, 2023. doi:10.1145/3579639.

[6] Red Hat. State of Kubernetes security report. https://www.redhat.com/en/resources/state-kubernetes-security-report, 2021. Accessed: 2021-06-01.

[7] B. Diarra, K. Guillouard, M. Ouzzif, P. Merle, and J.-B. Stefani. In-depth analysis of Kubernetes manifest verification tools for robust CNF deployment. In *27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, pages 17–24. IEEE, 2024. doi:10.1109/ICIN60470.2024.10494445.

[8] D. B. Bose, A. Rahman, and S. I. Shamim. 'under-reported' security defects in Kubernetes manifests. In *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, pages 9–12. IEEE, 2021. doi:10.1109/EnCyCriS52570.2021.00009.

[9] Canonical. Kubernetes and cloud native operations report 2021. https://juju.is/cloud-native-kubernetes-usage-report-2021, 2021. Accessed: 2025-03-23.

[10] D. Nag. Overcoming the Kubernetes skills gap with ChatGPT assistance. https://thenewstack.io/overcoming-the-kubernetes-skills-gap-with-chatgpt- 2023. Accessed: 2023-11-18.

[11] OpsCruise. Exploiting ChatGPT for trou-

bleshooting Kubernetes problems. https://www.opscruise.com/newsroom-post/exploiting-chatgpt-for-troubleshooting-kubernetes-problems, 2023. Accessed: 2025-04-04.

[12] R. Ravindranathan. ChatGPT for your Kubernetes cluster — k8sgpt. https://medium.com/techbeatly/chatgpt-for-your-kubernetes-cluster-k8sgpt-649f2cad1bd5, 2023. Accessed: 2023-11-19.

[13] M. Gupta, C. Akiri, K. Aryal, E. Parker, and L. Praharaj. From ChatGPT to ThreatGPT: Impact of generative AI in cybersecurity and privacy. *IEEE Access*, 11:80218–80245, 2023. doi:10.1109/ACCESS.2023.3300381.

[14] KubeLinter. Introduction — KubeLinter. https://docs.kubelinter.io/, 2024. Accessed: 2024-01-19.

[15] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman. XI commandments of Kubernetes security: A systematization of knowledge related to Kubernetes security practices. In *2020 IEEE Secure Development (SecDev)*, pages 58–64. IEEE, 2020. doi:10.1109/SecDev45635.2020.00025.

[16] A. S. Rahman, S. I. Shamim, D. B. Bose, and R. Pandita. Verifiability package for paper. https://figshare.com/s/bced7c8353853a983cd7, 2022. Accessed: 2022-08-20.

Nasim Tabibpour received her B.Sc. degree in Computer Engineering from Yazd University in 2021 and her M.Sc. degree in Computer Engineering, specializing in Secure Computing, from Isfahan University of Technology in 2025. Her master's thesis focused on the analysis of Kubernetes vulnerabilities and defensive countermeasures. Her research interests include cloud-native security, Kubernetes configuration analysis, DevSecOps, static analysis, and the security evaluation of human-written and AI-generated infrastructure configurations.

Mohammad Dakhilalian is a Professor in the Department of Electrical and Computer Engineering at Isfahan University of Technology (IUT). He received his B.Sc. (1989) and Ph.D. (1998) in Electrical Engineering from IUT, and his M.Sc. degree from Tarbiat Modarres University in 1993. Before joining IUT as a faculty member in 2001, he served as an Assistant Professor at the Faculty of Information and Communication Technology under the Ministry of ICT in Tehran from 1999 to 2001. During more than two decades at IUT, he advanced to full Professor. His current research focuses on cryptography and data security.