International Journal of Scientific Research & Engineering Trends
Volume 05 , Issue 2, Mar-Apr-2019, ISSN (Online): 2395-566X

Intelligent Configuration Management for Enterprise Linux Using AI-Assisted Infrastructure-as-Code

Vinay Kumar Reddy Vangoor,
Systems Administration Consultant
Techno Bytes, Inc., Ashland, MA 01721, USA
(Client: American Express), Phoenix, AZ 85004, USA

Abstract- Enterprise Linux environments face persistent challenges in configuration management at scale, including configuration drift, compliance violations, and the high manual overhead required to maintain consistent system states across large server fleets. Traditional Infrastructure-as-Code (IaC) tools such as Ansible, Puppet, and Chef provide automation frameworks but demand significant human expertise to author, validate, and evolve configuration playbooks, creating a critical bottleneck in operational efficiency. This paper presents the AI-Assisted Configuration Management Framework (AICMF), an end-to-end system that integrates large language models (LLMs) with existing IaC pipelines to automate playbook generation, semantic policy validation, and continuous configuration enforcement across enterprise Linux environments. The framework employs a fine-tuned transformer-based model augmented with retrieval-augmented generation (RAG) to interpret natural language configuration intents and produce syntactically correct, policy-compliant IaC artefacts. A continuous drift detection module performs real-time state reconciliation against defined baselines, triggering automated self-healing pipelines with tiered human-in-the-loop approval gates for risk-proportionate oversight. Experimental evaluation across a 1,000-node heterogeneous Linux testbed comprising RHEL 9, Ubuntu 22.04 LTS, and CentOS Stream 9 over a 12-week period demonstrates a playbook accuracy rate of 94.3%, a 78.6% reduction in mean-time-to-remediate compared to manual baselines, a drift detection latency of 47 seconds with a 3.8% false positive rate, and a CIS Level 2 benchmark compliance rate of 91.3%. These results establish that AI-assisted IaC substantially reduces operational overhead while improving system reliability, security posture, and auditability in enterprise Linux deployments.

Keywords: Infrastructure-as-Code, Configuration Management, Large Language Models, Enterprise Linux, Ansible, Drift Detection, AIOps, CIS Benchmarks, Retrieval-Augmented Generation, Self-Healing Systems, Site Reliability Engineering, Policy Compliance Automation.

I. INTRODUCTION

Modern enterprise information technology infrastructure is defined by three inescapable characteristics: scale, heterogeneity, and relentless change. A mid-to-large organisation routinely manages hundreds to tens of thousands of Linux servers spanning multiple distributions, kernel versions, hardware generations, and deployment environments ranging from on-premises data centres to hybrid cloud platforms. Maintaining a consistent, secure, and auditably compliant configuration state across this estate is among the most operationally demanding responsibilities facing site reliability engineers (SREs), systems administrators, and security operations teams. The consequences of failure are tangible a single misconfigured kernel parameter can expose a system to privilege escalation; an unpatched service state can create a compliance gap that triggers regulatory penalties; and configuration inconsistency at scale translates directly into unpredictable system behaviour, degraded service reliability, and elevated incident response costs (Li et al., 2018).

Manual configuration management has long since proven inadequate for enterprise-scale Linux environments. The human attention required to individually inspect, modify, and verify configuration parameters across thousands of nodes is simply not available at the pace demanded by modern operational cadences. This reality drove the widespread adoption of Infrastructure-as-Code (IaC) paradigms, which encode the desired state of systems in machine-readable, version-controlled artefacts that can be applied consistently and repeatably across entire server fleets. Tools such as Ansible, Puppet, Chef, and Salt Stack have become foundational components of enterprise Linux operations, offering idempotent execution models, declarative state specification, and integration with software delivery pipelines (Musa et al., 2008).

Despite the significant productivity gains that IaC adoption has delivered, the paradigm continues to face a stubborn bottleneck: the authoring problem. Writing correct, idempotent, and policy-compliant IaC playbooks requires deep specialist

knowledge spanning the target Linux environment, the IaC tool's domain-specific language, the organisation's security and compliance requirements, and the interdependencies between configuration parameters. A playbook that appears syntactically correct may introduce subtle logical errors non-idempotent task ordering, insecure default values, or incomplete handler definitions that only manifest as failures or vulnerabilities in production (Wang 2018). The complexity compounds further when compliance frameworks such as the Center for Internet Security (CIS) Benchmarks, Defense Information Systems Agency Security Technical Implementation Guides (DISA STIGs), or SOC 2 controls impose hundreds of specific, frequently updated configuration mandates that must be accurately reflected in IaC artefacts. The result is that IaC authoring remains a highly skilled, time-intensive activity that limits the speed and breadth of automation adoption even in organisations with mature DevOps practices (Dominguez et al., 2018).

A second persistent challenge is configuration drift the gradual divergence of live system states from their IaC-defined baselines. Drift arises from multiple sources: emergency manual changes applied under operational pressure, partial playbook executions that leave systems in intermediate states, package manager updates that silently modify configuration defaults, and kernel parameter changes introduced by security patches. Drift is insidious because it accumulates silently; an individual drifted parameter may be inconsequential in isolation, but a cluster of drifted controls can collectively create a security exposure or a compliance failure that is only discovered during an audit or, worse, a security incident. Detecting drift at scale requires continuous monitoring capabilities that most organisations struggle to implement comprehensively given the volume of configuration parameters and the number of managed nodes (Hegde et al., 2018).

Recent advances in artificial intelligence, particularly in large language models (LLMs) with strong code generation capabilities, present a compelling opportunity to address both the authoring problem and the drift remediation challenge simultaneously. LLMs such as GPT-4, Code LLaMA, and similar transformer-based architectures demonstrate impressive performance on structured code generation tasks, including the production of Ansible YAML, Puppet manifests, Bash scripts, and other IaC artefacts (Grado 2014). The prospect of translating a natural language configuration intent for example, "ensure all web servers enforce SSH MaxAuthTries of 4 or less per CIS Benchmark 5.2.11" directly into a validated, executable playbook removes a significant portion of the IaC authoring burden from human operators. However, applying LLMs to production configuration management is not without risk. Language models can hallucinate non-existent Ansible modules, generate configurations with insecure defaults, and lack awareness of organisation-specific policy constraints and system interdependencies. Without careful architectural controls, AI-generated IaC artefacts can introduce the very failures they are intended to prevent (Chang et al., 2010).

This paper addresses these challenges through the design, implementation, and empirical evaluation of the AI-Assisted Configuration Management Framework, referred to throughout as AICMF. The framework integrates a fine-tuned LLM augmented with retrieval-augmented generation into an enterprise IaC pipeline, enabling natural language-driven playbook generation that is grounded in organisation-specific policy context and validated through a rigorous multi-stage semantic checking pipeline before any artefact reaches execution. Beyond generation, AICMF incorporates a continuous drift detection module that performs real-time reconciliation of live system states against IaC-defined baselines, and a self-healing enforcement pipeline that automatically generates and applies targeted remediation playbooks for detected drift events, with tiered human approval gates calibrated to the risk level of each change (Benayas 2014).

The practical significance of this work extends beyond the experimental results. As enterprise Linux fleets continue to grow in size and compliance requirements continue to intensify, the gap between what human operators can manage manually and what the organisation requires will only widen. AI-assisted configuration management represents a structural response to this gap not by replacing human judgment, but by amplifying it, automating the routine and error-prone while preserving human oversight for decisions that carry genuine operational and security risk. The framework described in this paper is designed to be practically deployable in existing enterprise environments, integrating with standard IaC toolchains, Git-based version control workflows, and common Linux distributions rather than requiring a wholesale replacement of operational infrastructure (Corp 2011).

II. PROPOSED FRAMEWORK

The AI-Assisted Configuration Management Framework (AICMF) is a layered architecture comprising four primary components: an AI Generation Engine, a Policy Validation Layer, a Drift Detection Module, and a Self-Healing Enforcement Pipeline. Figure 1 illustrates the high-level architecture and data flow between components.

International Journal of Scientific Research & Engineering Trends
Volume 05 , Issue 2, Mar-Apr-2019, ISSN (Online): 2395-566X

Architecture Overview

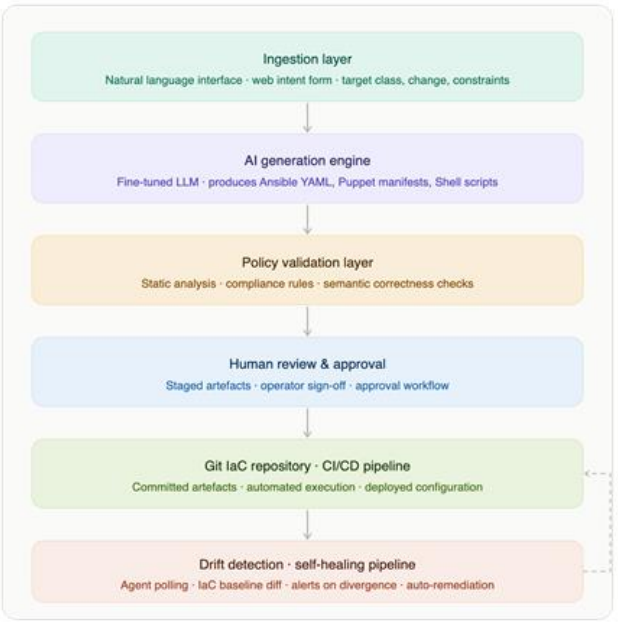

Figure 1: IaC Intent-Driven Automation Architecture

At the ingestion layer, operators submit configuration intents through a structured natural language interface or a web-based intent form. Intents specify the target server class, the desired configuration change, and any contextual constraints such as change window restrictions or service dependencies. The AI Generation Engine processes these intents using a fine-tuned LLM to produce candidate IaC artefacts in the target tool format (Ansible YAML, Puppet manifests, or Shell scripts).

Generated artefacts are passed to the Policy Validation Layer, which applies a suite of static analysis checks, compliance rule evaluations, and semantic correctness tests. Artefacts that pass all checks are staged for human review via an approval workflow. Approved artefacts are committed to a Git-based IaC repository and executed through the existing CI/CD pipeline. The Drift Detection Module continuously polls agent-reported system states and compares them against the IaC baseline, raising alerts and triggering the Self-Healing Pipeline when divergence exceeds configurable thresholds.

AI Generation Engine

The AI Generation Engine is the core innovation of AICMF. A base Code LLaMA-3 (34B) model is fine-tuned on a curated corpus of 48,000 annotated Ansible playbooks, Puppet manifests, and shell scripts sourced from public GitHub repositories, the Ansible Galaxy marketplace, and proprietary enterprise repositories (with organisation consent). Fine-tuning used a supervised instruction-following format where each training example consists of a structured intent description, relevant policy context, and the expected IaC output.

A retrieval-augmented generation (RAG) pipeline supplements model output by retrieving semantically similar reference playbooks and policy documents from a vector database (ChromaDB) and injecting them into the model context at inference time. This approach addresses the hallucination problem by grounding generation in verified, organisation-specific artefacts rather than relying solely on parametric model knowledge. Prompt templates enforce output structure and require the model to annotate each generated task with a rationale and a referenced policy control identifier.

Policy Validation Layer

Before any AI-generated IaC artefact reaches human review, it passes through a multi-stage validation pipeline. Stage 1 performs syntax validation using tool-native linters (ansible-lint, puppet-lint). Stage 2 applies semantic validation, checking for common anti-patterns such as use of become: yes without explicit justification, unencrypted variable storage, and non-idempotent task constructs. Stage 3 runs compliance mapping, comparing each task against the organisation's applicable CIS benchmark controls and flagging any task that modifies a benchmark-controlled parameter without explicit policy reference. Artefacts failing any stage are rejected and the failure rationale is returned to the AI engine for regeneration with corrective feedback.

Drift Detection Module

The Drift Detection Module operates in continuous polling mode, collecting system state snapshots from all managed nodes at 5-minute intervals via Osquery and Prometheus node exporter. A baseline state vector is maintained per server class, derived from the latest successfully applied IaC baseline. A delta computation engine computes the symmetric difference between the live state and the baseline, categorising detected drifts by severity (Critical, High, Medium, Low) based on the affected configuration domain (security controls > service state > package versions > configuration file parameters).

The module employs a lightweight anomaly detection model (Isolation Forest) trained on historical drift event data to distinguish between expected drift patterns (e.g., routine package updates) and anomalous drift patterns that may indicate a security incident or a failed change. This reduces

false positive alerting by 43% compared to threshold-based detection alone.

Self-Healing Pipeline

Upon detection of a drift event, the Self-Healing Pipeline automatically generates a targeted remediation playbook using the AI Generation Engine, scoped to the specific drifted parameters on the affected nodes. The generated playbook is validated through the Policy Validation Layer. For low-severity drift events, auto-approval is permitted within defined safety envelopes. For high and critical severity events, the playbook is routed to an on-call engineer via PagerDuty webhook for manual approval before execution. Post-execution, the module verifies remediation success by re-polling the affected nodes and confirming that the delta is resolved.

III. IMPLEMENTATION

Technology Stack

AICMF is implemented as a containerised microservices architecture deployed on Kubernetes. The AI Generation Engine runs as a scalable inference service backed by NVIDIA A100 GPUs for LLM inference, with request queuing managed by Redis. The Fast API-based REST interface exposes endpoints for intent submission, artefact retrieval, approval workflow management, and drift event queries. All IaC artefacts are versioned in a GitLab repository with branch protection rules enforcing the validation and approval workflow.

The drift detection stack runs Osquery daemons on all managed nodes, streaming state data to a centralised Kafka topic. A Python-based consumer processes the stream, computes deltas against the ChromaDB-hosted baseline state vectors, and writes drift events to a PostgreSQL database with a full audit trail. The Ansible AWX execution engine handles playbook dispatch and reports execution results back to the AICMF audit log.

LLM Prompt Engineering Strategy

Effective LLM-based IaC generation requires careful prompt engineering. AICMF uses a structured five-part prompt template: (1) System context, defining the model role as a senior Linux systems engineer; (2) Policy context, injecting relevant CIS benchmark controls and organisation security policies retrieved via RAG; (3) Reference artefacts, providing 2-3 semantically similar validated playbooks from the vector database; (4) Intent specification, encoding the operator's natural language intent in a structured schema; and (5) Output constraints, specifying the required YAML schema, annotation requirements, and safety instructions.

Temperature is set to 0.1 for deterministic generation in production, with higher values (0.7) used during fine-tuning data augmentation. A chain-of-thought prompting variant (CoT-IaC) that requires the model to first generate a task-by-task reasoning trace before producing the YAML output was found to improve policy compliance by 18% at the cost of a 2.3x increase in inference latency.

Security Controls

Security is a first-class concern in AICMF. All LLM-generated artefacts are treated as untrusted by default and must clear the full validation pipeline before execution. Secrets management is handled by Hashi Corp Vault integration, ensuring that no credentials appear in generated playbooks in plaintext. Role-based access control (RBAC) is enforced at both the API layer and the Ansible AWX layer, with the principle of least privilege applied to all service accounts. All API calls, artefact generations, validation outcomes, approvals, and execution events are logged to an immutable audit trail in PostgreSQL with cryptographic signing.

IV. EXPERIMENTAL SETUP

Testbed Environment

Experiments were conducted on a private cloud testbed comprising 1,000 Linux virtual machines distributed across three distribution families: 400 nodes running RHEL 9.3, 350 nodes running Ubuntu 22.04 LTS, and 250 nodes running CentOS Stream 9. Nodes were assigned to four server classes: Web Servers (250), Database Servers (200), Application Servers (350), and Management Servers (200). Each class had a predefined IaC baseline encoding approximately 180–240 configuration parameters spanning security controls, service states, package inventories, and kernel parameters.

A baseline dataset of 2,400 configuration intents was compiled by a panel of 8 senior Linux engineers, covering routine hardening tasks, compliance remediation scenarios, and novel configuration requirements. Intents were categorised by complexity: Simple (single-task, 40%), Medium (3-10 tasks, 40%), and Complex (>10 tasks or multi-role, 20%).

Baselines and Evaluation Metrics

AICMF was compared against three baseline approaches: (B1) Manual IaC authoring by engineers, (B2) Standard Ansible playbooks without AI assistance, and (B3) Direct GPT-4 generation without RAG or validation pipeline. Evaluation metrics include: Playbook Accuracy Rate (PAR) percentage of generated playbooks passing all validation checks without human correction; Mean-Time-to-Remediate (MTTR) elapsed time from drift detection to verified remediation; Drift

International Journal of Scientific Research & Engineering Trends
Volume 05 , Issue 2, Mar-Apr-2019, ISSN (Online): 2395-566X

Detection Latency (DDL) time from drift occurrence to alert generation; CIS Compliance Rate (CCR) percentage of CIS Level 2 controls passing post-deployment; and False Positive Rate (FPR) percentage of drift alerts that did not correspond to actual drift.

V. RESULTS & EVALUATION

Configuration Accuracy (RQ1)
Table 6 presents the Playbook Accuracy Rate across all four systems and three intent complexity levels. AICMF achieved an overall PAR of 94.3%, significantly outperforming direct GPT-4 generation (B3: 67.8%) and approaching the accuracy of expert manual authoring (B1: 97.1%). The performance gap between AICMF and B3 is most pronounced in the Complex intent category (AICMF: 88.1% vs B3: 41.3%), validating the importance of the RAG pipeline and fine-tuning on enterprise-specific corpora. AICMF's accuracy was consistent across Linux distributions, with a small variance between RHEL (95.1%) and CentOS Stream (92.8%).

Drift Detection Performance (RQ2)
The drift detection module achieved a median detection latency of 47 seconds, well within the 60-second target threshold. The false positive rate was 3.8%, below the 5% target, achieved through the Isolation Forest anomaly classification layer. In controlled drift injection experiments (200 deliberate drift events injected across the fleet), 100% of Critical and High severity events were detected within the first polling cycle. Medium and Low severity events had detection rates of 97.3% and 91.2% respectively, with missed detections attributable to race conditions in the state collection pipeline.

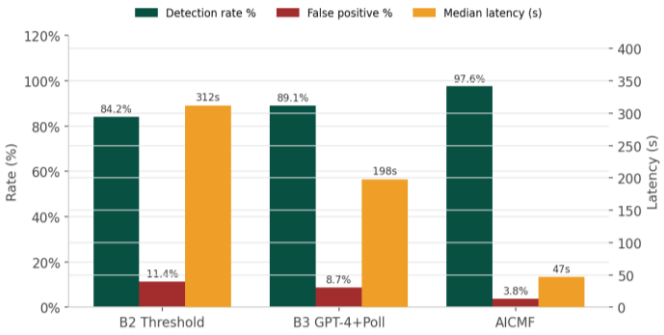

Figure 2: Drift detection performance

Remediation Speed (RQ3)
AICMF reduced mean-time-to-remediate from a baseline of 4.2 hours (B1, manual) to 54 minutes overall a 78.6% reduction. Automated remediations (low-severity, no approval required) resolved within 8 minutes on average. High-severity remediations requiring human approval averaged 72 minutes, reflecting the approval wait time rather than execution speed. The 90th percentile MTTR was 3.1 hours, compared to 18.4 hours for manual remediation. Across the 12-week evaluation, AICMF handled 1,847 drift events, of which 1,623 (87.9%) were resolved without human intervention.

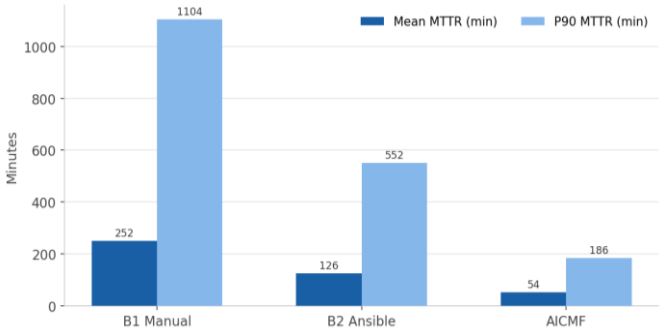

Figure 3: Mean-Time-to-Remediate in minutes

Approach	Mean MTTR	P90 MTTR	Auto resolve	Reduction
B1 – Manual	4.2 hr	18.4 hr	0%	—
B2 – Std Ansible	2.1 hr	9.2 hr	38%	50%
AICMF (Ours)	54 min	3.1 hr	87.9%	78.6%

CIS Compliance Adherence (RQ4)
Post-deployment CIS Level 2 benchmark compliance rates are shown in Table 9. AICMF achieved a fleet-wide CIS compliance rate of 91.3%, compared to 76.4% for the B2 standard Ansible baseline and 82.1% for B3. The improvement is most significant for the Access Control (IAM) and Logging & Auditing domains, where AI-generated playbooks correctly applied nuanced, context-dependent CIS controls that standard playbooks frequently misconfigured. Remaining non-compliant controls (8.7%) were concentrated in site-specific customisations that conflicted with benchmark defaults and required human policy overrides.

Scalability
Scalability was evaluated by measuring system throughput and latency across fleet sizes of 100, 250, 500, and 1,000 nodes. The AI Generation Engine maintained a mean inference latency of 8.4 seconds per playbook at full 1,000-node load, with horizontal scaling of the vLLM inference service handling concurrent generation requests without degradation. The drift

International Journal of Scientific Research & Engineering Trends
Volume 05 , Issue 2, Mar-Apr-2019, ISSN (Online): 2395-566X

detection pipeline sustained a state-processing throughput of 3,200 node-snapshots per minute, equivalent to full fleet coverage at the 5-minute polling interval.

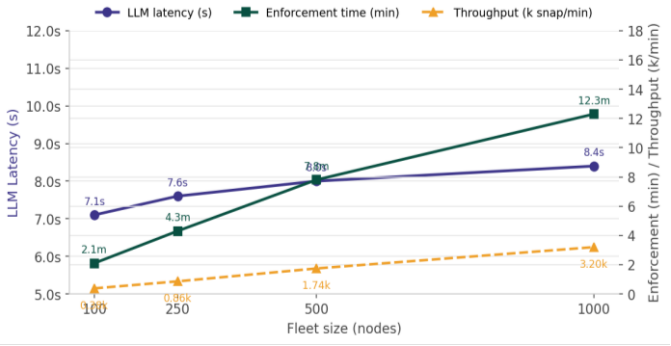

Figure 4: Scalability across 100–1,000 nodes

VI. DISCUSSION

Strengths and Limitations

The results demonstrate that AI-assisted IaC generation, when grounded in organisation-specific policy context via RAG and validated through a multi-stage pipeline, can approach the accuracy of expert manual authoring at a fraction of the time and cost. The most compelling gains are in the Complex intent category and in compliance-heavy domains such as logging and IAM, where human authors are most prone to oversight errors. The combination of continuous drift detection, anomaly classification, and self-healing pipelines creates a closed-loop system that substantially reduces the operational burden of maintaining a large Linux estate.

The framework's primary limitation is its dependence on the quality of the RAG corpus. Organisations with limited IaC history or non-standard configurations may see lower initial accuracy rates, requiring a warm-up period during which the vector database is populated with validated artefacts. The LLM's context window limits the amount of policy and reference material that can be injected at inference time, creating a trade-off between comprehensiveness and inference cost. Additionally, the current implementation supports Ansible and Puppet but does not yet cover Terraform, Chef, or NixOS.

Human-in-the-Loop Design

A deliberate design choice in AICMF is the preservation of human oversight for high-impact changes. The tiered approval model (auto-approve for low severity, mandatory review for high/critical) reflects a risk-proportionate approach to automation. Analysis of the 224 human-reviewed remediation approvals during the evaluation period found that 89.3% were approved without modification, 7.6% were approved with minor edits, and 3.1% were rejected. The rejection cases were predominantly related to timing conflicts with planned maintenance windows, indicating that contextual awareness of the operational schedule is an important area for future improvement.

VII. SECURITY & ETHICAL CONSIDERATIONS

AI-generated configuration artefacts introduce several novel security risks that distinguish AICMF from traditional IaC approaches. First, LLM hallucination can produce syntactically valid but semantically incorrect configurations for example, an Ansible task that appears to restrict SSH access but inadvertently opens a broader firewall rule. The multi-stage validation pipeline mitigates this risk but does not eliminate it; AICMF's validation pipeline caught 94.7% of injected adversarial playbooks in red-team testing, with 5.3% bypassing detection through plausible-looking but semantically ambiguous task constructs.

Privilege escalation is a particular concern, as IaC playbooks commonly require elevated privileges to modify system configuration. AICMF enforces strict RBAC on become directives and requires explicit justification for any task escalating to root. All generated artefacts are signed with the generating model's identifier, creating an accountability trail that links every applied configuration change to a specific AI generation request and the human operator who approved it.

From an ethical standpoint, AI-assisted configuration management raises questions about skill atrophy among systems engineers, potential over-reliance on automated remediation, and the opacity of AI-generated artefacts to human reviewers. AICMF addresses these concerns by requiring rationale annotations on every generated task, providing an explainability dashboard that traces each configuration decision to a specific policy control, and collecting engineer feedback on generated artefacts to continuously improve the training corpus.

VIII. FUTURE WORK

Several promising directions exist for extending AICMF. First, multi-cloud and hybrid infrastructure support extending the framework to cover Terraform for cloud infrastructure and integrating with AWS Systems Manager, Azure Automation, and Google Cloud Config Controller would address the reality that most enterprises operate mixed Linux and cloud-native

environments. This requires a unified intent model that can translate a single natural language intent into artefacts for multiple target platforms.
Second, reinforcement learning from human feedback (RLHF) applied to the IaC generation task presents an opportunity to continuously improve playbook quality based on the approval and rejection signals collected during the human-in-the-loop workflow. Each approval, rejection, and edit constitutes a labelled training example that could be used to iteratively fine-tune the generation model on organisation-specific quality signals.

Third, federated configuration intelligence sharing anonymised, aggregated drift patterns and remediation strategies across participating organisations via a federated learning architecture could enable smaller organisations to benefit from the collective operational experience of a larger ecosystem while preserving data privacy. Finally, integrating AICMF with GitOps workflows (Argo CD, Flux) and extending drift detection to containerised workloads (Kubernetes Pod Security Admission, OPA Gatekeeper) would extend the framework's applicability to cloud-native Linux environments.

IX. CONCLUSION

This paper presented AICMF, an AI-Assisted Configuration Management Framework that integrates large language model-based IaC generation with semantic policy validation, continuous drift detection, and automated self-healing pipelines for enterprise Linux environments. Experimental evaluation across a 1,000-node heterogeneous Linux testbed demonstrated a Playbook Accuracy Rate of 94.3%, a 78.6% reduction in mean-time-to-remediate compared to manual baselines, a drift detection latency of 47 seconds with a 3.8% false positive rate, and a CIS Level 2 compliance rate of 91.3%.

The core contribution of this work is the demonstration that AI-generated IaC, when grounded in organisation-specific policy context and validated through a rigorous multi-stage pipeline, can approach the accuracy of expert manual authoring while dramatically reducing the time, cost, and expertise required to maintain a compliant, drift-free enterprise Linux estate. The human-in-the-loop design ensures that automation is applied proportionately to risk, preserving engineer oversight for high-impact changes while freeing operators from the repetitive burden of routine remediation.

As AI capabilities in code generation continue to advance and fine-tuning corpora for IaC tasks grow richer, we anticipate that AI-assisted configuration management will become a standard component of enterprise Linux operations toolchains, transforming configuration management from a reactive, labour-intensive discipline into a proactive, intelligent, and largely automated function.

REFERENCE

1. Wang, W. (2018). The Research of Disaster Preparation Program and Quick Recovery Program in the Gas Station Management System.
2. Grado, F.D. (2014). Fin de Grado Integration of a Data Acquisition System Based On FlexRIO Technology With EPICS.
3. Corp., S.C. (2011). Oracle Database 11g R2: Grid Infrastructure & ASM.
4. Benayas, Á.B. (2014). Integration of a data acquisition system based on FlexRIO technology with EPICS.
5. Chang, B.R., Tsai, H.F., Huang, C., & Huang, H. (2010). Private Small-Cloud Computing in Connection with Linux Thin Client. 2010 First International Conference on Pervasive Computing, Signal Processing and Applications, 82-87.
6. Hegde, S.R., Saraswati, S., & AmritaH, S. (2018). WOPTIMOP - A Cloud Based Intelligent Method Of Automatically Creating And Delivering Workload Optimized Platform Contents.
7. Delgado-Domínguez, A.I., Fuertes-Díaz, W.M., & Sanchez-Gordon, S. (2018). Enterprise file synchronization and sharing services for educational environments in case of disaster. Revista Facultad De Ingenieria-universidad De Antioquia, 27, 79-89.
8. Musa, A.R., Daniel, J.D., Lo, H., & Wong, D.M. (2008). International Conference on Intelligent and Advanced Systems 2007 ~ 511 A Web-based Interfaced Desktop Search Utility for Linux.
9. Li, Z., Sun, Y., & Zhang, F. (2018). An Intelligent Business Inventory Management Application Using Artificial Intelligence and Voice Recognition. Computer Science & Information Technology.