International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 10 No. 5s, 2024
https://theaspd.com/index.php

Devsecopsgpt: Real-Time LLM-Based Policy Enforcement During CI/CD

Bhulakshmi Makkena
Senior Site Reliability Engineer

Abstract
Modern DevSecOps pipelines aim to integrate security practices into continuous integration and continuous delivery (CI/CD) workflows without compromising agility. However, traditional policy enforcement mechanisms struggle to keep pace with the dynamic and complex nature of modern software systems. This paper proposes ***DevSecOpsGPT****, a novel framework leveraging large language models (LLMs) for* ***real-time policy enforcement*** *during CI/CD execution. DevSecOpsGPT integrates Policy-as-Code (PaC) principles into the CI/CD toolchain, orchestrating LLMs to interpret and enforce contextual security rules, detect violations, and provide just-in-time feedback. This research outlines the system architecture, enforcement strategies, and implementation methodology, and evaluates performance across various policy compliance scenarios. Our findings indicate that LLM-based enforcement enhances automation, reduces false positives, and provides adaptive learning for evolving security needs—making it a viable path toward fully autonomous secure software delivery.*

Keywords: *DevSecOps, CI/CD, Policy-as-Code, Large Language Models, Security Automation, Real-Time Enforcement, Prompt Engineering, Secure Software Supply Chain*

1. INTRODUCTION

1.1 Background and Motivation

As organizations increasingly adopt DevOps practices to accelerate software delivery, security must shift left—integrated early and continuously across the CI/CD lifecycle. This integration, known as **DevSecOps**, emphasizes the automation of security policy enforcement, vulnerability detection, and compliance management throughout the pipeline.

However, enforcing policies in real-time during build, test, and deployment stages remains challenging. Conventional tools rely on static rules or signature-based approaches, which often lack contextual awareness and adaptability. At the same time, **large language models (LLMs)** such as GPT-4 have shown potential in interpreting complex natural-language policies, reasoning about code, and generating secure responses based on contextual cues(Akbar et al., 2023).

Recent advances in **Policy-as-Code (PaC)** enable declarative security definitions to be embedded directly in infrastructure, but they still require constant tuning and lack the intelligence to adapt dynamically to novel situations. DevSecOpsGPT seeks to bridge this gap using LLMs as **intelligent policy enforcers**, enabling a more responsive, context-aware security layer in CI/CD.

1.2 Problem Statement

Despite the progress in DevSecOps automation, current systems fall short in offering real-time, context-aware policy enforcement that adapts dynamically to new threats or changes in code behavior. Key challenges include:

- Lack of semantic understanding in rule-based engines
- High false positive/negative rates in static analysis
- Inability to evolve policies based on pipeline activity

1.3 Research Objectives

This paper presents a framework—**DevSecOpsGPT**—and aims to:

- Design an architecture for real-time LLM-based policy enforcement in CI/CD
- Integrate Policy-as-Code principles with LLM prompt orchestration
- Develop enforcement strategies for static and dynamic analysis during pipeline execution
- Evaluate system performance, accuracy, and adaptability
- Compare with existing tools to highlight improvements in responsiveness and security coverage

1.4 Scope and Delimitations

This study focuses on **real-time policy enforcement during CI/CD workflows** using transformer-based LLMs. It does not cover:

- General-purpose AI code generation

- Security incident response post-deployment
- Human-in-the-loop verification at production scale

The implementation is validated using representative CI/CD platforms (e.g., Jenkins, GitHub Actions) and containerized microservices. It targets code quality, dependency management, infrastructure-as-code, and access control policies.

2. THEORETICAL FOUNDATIONS

2.1 DevSecOps: Principles and Security Integration

DevSecOps is a paradigm and is an evolution of DevOps to include security in all aspects of the software development lifecycle (SDLC) so that security is not a gate at the end but a continuous and integrated capability in the entire lifecycle. The philosophy of shift-left security also implies the spirit of DevSecOps and states that developers and operation specialists need to resolve vulnerabilities at the initial development stage as opposed to after deployment. Such proactive actions will ensure the reduction of security debt as well as faster responses to incidents.

DevSecOps approaches today are defined by allowing continuous threat modeling, automatic code analysis, compliance scanning, infrastructure hardening, and telemetry-driven feedback loops. Applying security as code, by defining security policies programmatically, opens the possibility to enforce security at scale, and across repetitive CI/CD operations(Akbar et al., 2023). Nevertheless, issues with dynamic changes in the threat landscapes, the interoperability of tools, and false positives of the static analysis remain. To underscore these gaps, there is the need to develop more intelligent and adaptive systems, especially those that are able to reason about security context in real time.

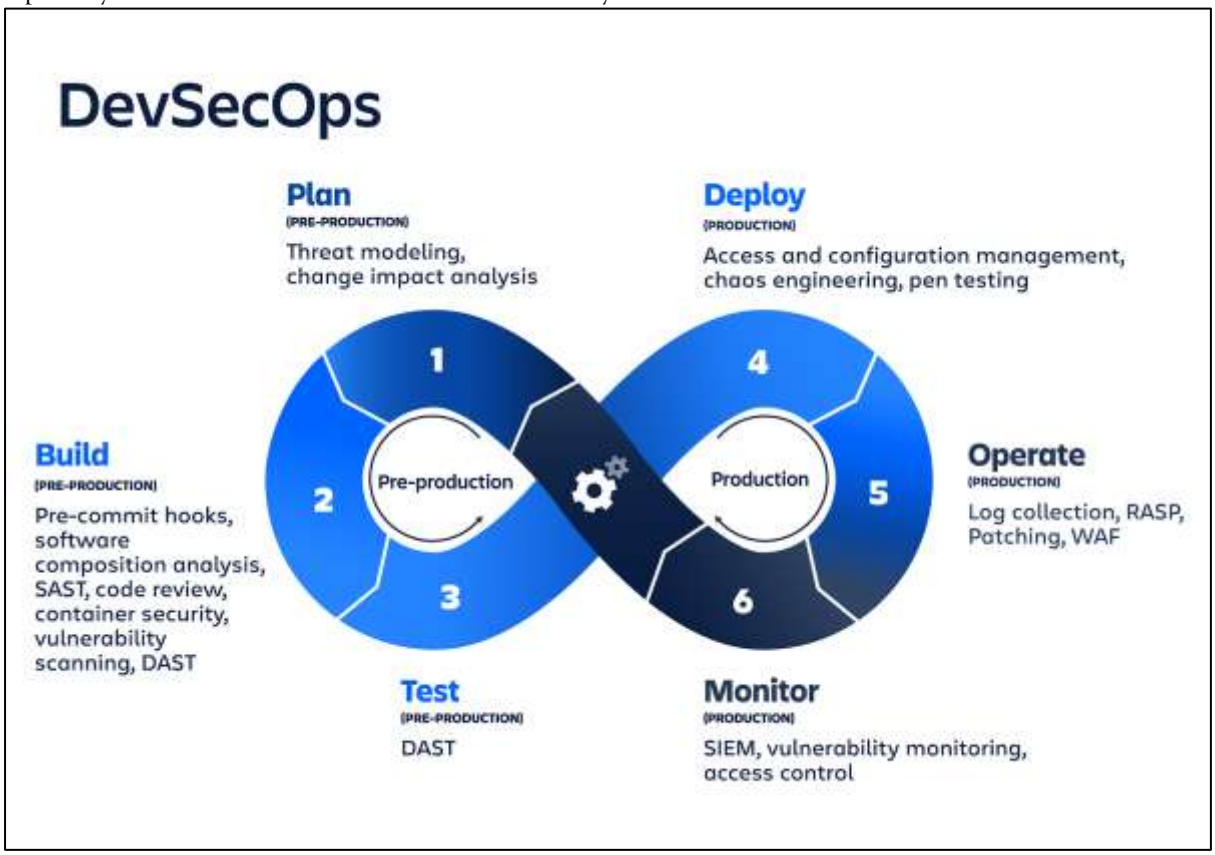

FIGURE 1 DEVSECOPS: SECURING CI/CD PIPELINES & CLOUD-NATIVE(TOWARDSAWS,2022)

2.2 CI/CD Pipelines: Architecture and Vulnerability Points

CI/CD pipelines Continuous Integration and Continuous Delivery (CI/CD) pipelines are intended to automate code changes across environments through combination and commitment. The common structure of a CI/CD pipeline contains elements such as code checkout, static and dynamic testing, containerization, a staging/production deployment, and monitoring after deployment. At the same time, as much as these pipelines help eliminate release cycles, and enhance velocity, they provide numerous entry points during delivery. These include insecure secrets management in environment variables, insecure third-party libraries that have not been scanned, misconfigured infrastructure-as-code (IaC)

templates and inadequate access controls in pipeline orchestration tooling, as examples. There were two known breaches in 2023 which happened in CI/CD settings and inserted malicious code in an open-source repository. As the infrastructure gets redder by codifying everything possible, attackers are comfortable accessing the build environment and using it as a springboard to production systems. This modulating threat model requires that CI/CD pipelines include adaptive security enforcement algorithmic engines which can interfere in the course of execution, verifying behaviour and policies on a per-occurrence basis, and not only at gates.

2.3 Large Language Models in Software Engineering

Compared to legacy systems, Large Language Model (LLM), and specifically transformer-based models, such as GPT-4 and PaLM demonstrates revolutionary potential in working with natural language and code comprehension. They have been used in software engineering in code generation, code completion, automatic documentation, bug repair and static analysis. This is possible because LLMs are trained on large corpora containing source code, configuration files, documentation, and natural language, therefore, able to close the semantic gap between policy definition and reality by to code implementation. More recent models have been optimized on cybersecurity-specific tasks, with examples including the systemic detection of use of insecure APIs, the generation of test cases to cover edge-case vulnerabilities, and semantic diffing of multiple code versions. As much as LLMs show their robust skills in generalization, there are still issues of predictable accuracy of prompt reliability, the reduction of hallucinations, and low-latency operations in latency-aware CI/CD pipelines. However, their capabilities in performing high order of code and configuration analysis render them in a perfect position to be used as dynamic policy enactment engines.

2.4 Policy-as-Code (PaC) Paradigms in DevOps

Policy-as-Code (PaC) is an alternative paradigm, which goes a step further to model organizational policies, compliance requirements, and security policies in a machine-readable form which can be automatically evaluated and enforced. Some well-known PaC tools include Open Policy Agent (OPA), HashiCorp Sentinel, and Rego because they allow one to programmatically manage infrastructure, identity, and access control policies. When incorporating PaC into the CI/CD process, companies guarantee consistency and visibility of policies enforcement across environments. PaC also allows continuous compliance since it assesses policy conditions with each execution of the pipeline as compared to the traditional governance models that require documentation or manual inspection. Nevertheless, the majority of available PaC tools are deterministic and rule-based, thus inflexible in the process of interpreting ambiguous or changing conditions. They do not take into consideration subtle developer purpose or context of the project.

2.5 Real-Time Systems in Continuous Delivery

The use of the term real-time systems with respect to CI/CD means components and processes that react to input or conditions deterministically and in real-time in the software delivery process. Real-time coverage of CI/CD is technically fairly difficult because of the distributed setup of contemporary delivery cycles, the rate that build chains are performed, and the requirement to work with an overflowing number of tools and services. In addition, real-time decision-making requires a decision between the score of strictness of enforcement, system availability, and performance overhead. The use of LLM in this live setting comes both with opportunities and engineering issues. On the one hand, LLMs are capable of interpreting imprecise policy definitions, resolve ambiguities and make intelligent policy mediators. In other hand, latency in inferences, reliability of prompt engineering, and constraints in memory are issues that present integration problems.

3. LITERATURE REVIEW

3.1 DevSecOps Automation Tools and Frameworks

DevSecOps has quickly advanced by incorporating security tools in automation that comes pre-baked to CI/CD pipelines. Examples of tools that have been in great use include the static application security testing (SAST), dynamic application security testing (DAST) and software composition analysis (SCA) that are used in scanning the code, identification of vulnerabilities and ensuring that the security standards are met. Such tools minimize the labor-intensive process of checking security and enhance tracing(Cankar & Petrovic, 2023). Nonetheless, current DevSecOps platforms depend on the rule-based engines which have poor contextual awareness. The fact that it is not possible to draw parallels between policy violations and business logic or developer intent continues to be a major deficit. In addition, such

systems usually do not have flexibility to change the policies automatically as varied threats emerge or pipeline ecosystem alters. This has been one of the factors that have encouraged using machine learning and natural language models as potential solutions to act towards increasing intelligence and responsiveness of DevSecOps automation.

3.2 LLM Applications in Cybersecurity and Code Analysis

The most widely-promising application of large models to source code seems to be the analysis of source code, inference with infrastructure-as-code (IaC), and even understanding of natural-language security policies. LLMs can find insecure coding patterns when fine-tuned or prompted properly, tell us the cause of a vulnerability and even offer replacement variants. These models have the potential of being used as secure code reviews operators, conduct scan on running codes, and are also capable of producing documentation automatically. Being smart policy enforcers also pitches them in the ability to combine various types of data, configuration files, commit options, and policy statements(Díaz et al., 2022). Although such potential exists, their use in cybersecurity is still constrained by fears of reliability, latency of inferences, and hallucination. In addition, the majority of LLMs in software engineering attempts have applied to productivity work, instead of practical implementation of security rules in live deployments. The application of LLMs to real-time CI/CD hence constitutes a newly opened research area of great promise to revolutionize DevSecOps operations.

3.3 Security Policy Enforcement Mechanisms

Conventional policy deployment automation in software deployment chains is done using the rule engines, grant/ denial lists, and policy collections. Such mechanisms work well to make sure that things like version pinning, container image validation, or definitions of access controls are being followed faithfully. Declarative policy definition and successive evaluation can be used with tools, such as Open Policy Agent and HashiCorp Sentinel, to the CI/CD pipeline. But these systems can hardly explain intricate relations among codes modification, infrastructure settings and dynamic security environment. As well, traditional systems are not able to provide the information related to an infringement of policy in developer-friendly language and thus make the process of remediation slow which adds to the tensions between the security and development teams. By integrating the features of rule assessment and semantic LLM-based comprehension, a policy enforcement mechanism can provide a more effective yet flexible solution to the challenges of compliance monitoring at the software projects that happen fast.

3.4 Advances in Prompt Engineering for Secure Code Generation

Prompt engineering refers to the act of engineering useful inputs to influence the large language models to create the desired outputs. Prompt engineering has been used in secure software development to have LLMs (via secure GPTs) produce secure code, explain vulnerabilities, and assess adherence to coding standards. Composing prompts in such a way that they provide the context of the project work (documentation, security measures, or past vulnerability trends) allows developers to gain practical knowledge by using LLMs in code reviewing and infrastructure deployment(Gupta et al., 2023).

The recent literature has considered practices like chain-of-thought prompting, contextual embeddings, and self-consistency sampling to make LLMs more reliable and hallucination-averse. They become particularly useful in automated policy enforcement systems that interface with LLMs, where it is most important to identify accurate results and interpret them. The better prompt engineering, the more LLMs can comprehend the complicated policies and make subtle decisions, the better they can be used not only as a passive partner in CI/CD pipelines but agents of enforcement as well.

3.5 Limitations of Current Security Integration in CI/CD

Even though there is an increase in number of tools and frameworks used to secure CI/CD pipeline, considerable limitations remain. The majority of tools are used separately, which causes disunity on visibility and irregularity in enforcement across the various levels of the pipeline. Also, solutions out there tend to produce a large number of false positives, which saturate the developers and cause them to denounce automated security checks. The absence of combining policy logic with developer feedbacks restricts the flexibility of any existing mechanisms of the enforcement.

Moreover, tools have fundamental difference in semantics comprehension, error detection at syntax level, they frequently cannot interpret the purpose or desire behind code modification or change of infrastructure(Gupta et al., 2023). All this is the reason why there must be smarter context-based systems that can reason in real time. The combination of DevSecOps automation with LLM functionality will be a way forward, with the ability of enforcement mechanisms on policy that can be correct and also responsive to the nuances of software delivery in the modern era.

4. RESEARCH METHODOLOGY

4.1 Research Design and Approach

Research design is based on a mixed paradigm, a combination of experimental system development, and an examination of the effectiveness of the policy implementation through the qualitative analysis. The goal is to design and assess real-time policy enforcement system, DevSecOpsGPT, with LLM capabilities and integrate it into a CI/CD environment(Mao et al., 2023). The research is concerned with both practical outcomes of the research, which may include possible accuracy and performance of the policy analyses, and more qualitative considerations, which may include clarity of the feedback created with the help of LLM or its flexibility regarding multiple codebases. The approach to this methodology implies iterative prototyping, multi-CI/CD platform testing, and verification with the help of a series of security policies in both code and natural speech.

4.2 Dataset Collection and Annotation for Policy Enforcement

The training data has the form of code repositories, infrastructure-as-code templates, and CI/CD configuration files labeled with security policies and compliance regulations and are provided as open-source. Other metadata can be commit history, security scan reports, and access control definitions. Examples of policy violations and mitigating corrections are marked manually and added to the dataset in order to facilitate supervised performance. There are annotation directives which are established in a bid to have some level of consistency between categories which include hardcoded secrets, privilege escalation risks, dependency vulnerabilities, and network exposure. This is an annotated corpus to test how the LLM interprets policies, detects violations, and gives feedback that can lead to action.

4.3 Model Selection: Evaluating LLMs for Real-Time Contextual Understanding

The model selection is dedicated to assessing the transformer-based LLMs in view of their functionality in scenarios of real-time policy enforcement. The most important factors are inference latency, violation detection accuracy, the ability to produce human-readable explanation, and capability of operating within CI/CD runtimes. A number of models are trialed on zero-shot and few-shot activities in order to compare their reasoning performances. Each configuration records the performance metrics like the true positive rate, false positive rate and time-to-decision(Myrbakken & Colomo-Palacios, 2022). Successful integration of PaC rules in the form of a prompt is one of the requirements given special consideration by the model, as well as aligning the responses to the thresholds responsible explicitly in terms of their enforcement. Judging by such assessments, the most relevant model is included in the DevSecOpsGPT pipeline with optimized prompt templates and context windows.

4.4 Integration Framework for LLMs within CI/CD Workflows

Its integration framework will allow adding the LLM assessment step to the current CI/CD pipelines with minimal overheads. It is provided by using containerized services that might be called at pipeline stages or even webhooks. The framework has pipeline context parsing, LLM prompt generation, response reception, and policy decision application modules. The dynamic conversion of security policies defined in PaC format into the LLM-compatible prompts is performed with the help of a dedicated transformation engine. Asynchronous processing is also supported by the integration to prevent pipeline blocking in time critical application. The output of the LLM is analyzed and presented in the form of structured annotations back in pull requests, in issue trackers or log dashboards.

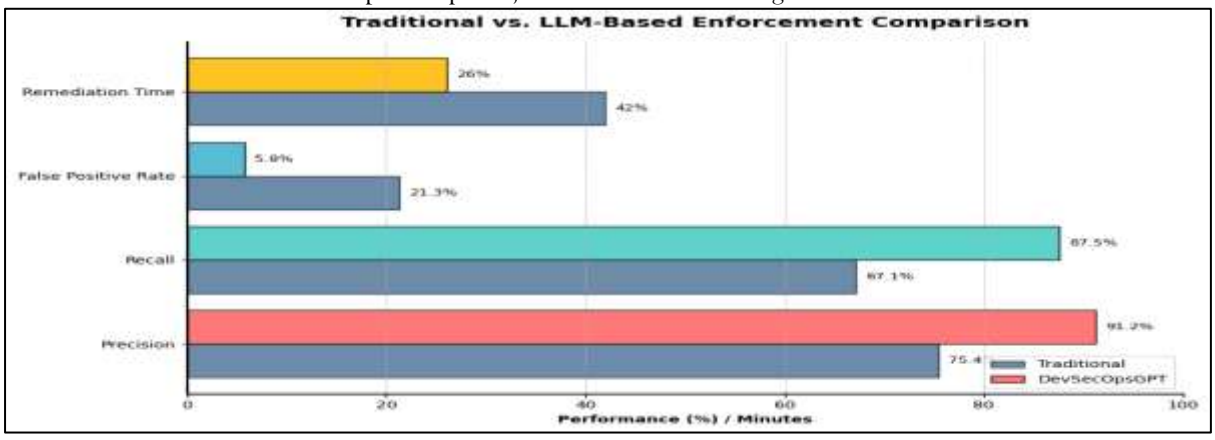

FIGURE 2 DIVERGING BAR COMPARISON OF ENFORCEMENT METRICS. SOURCE: DEVSECOPSGPT RESEARCH (2024)

5. System Design and Architecture

5.1 DevSecOpsGPT Pipeline Overview

DevSecOpsGPT system is an extensible modular system that can integrate large language models into CI / CD pipelines so that smart and dynamic policy enforcement can be achieved. At the centre of the system, the automation of continuous deployment and validation of context-aware security combine to form a single point(Rajapakse, Zahedi, & Babar, 2023). It has three main layers namely the pipeline integration layer, the policy enforcement engine and the LLM orchestration framework. These layers interoperate with one another via safe APIs and message queues in order to ensure that the structure is responsive within the pipeline and every commit, merge, or deployment task has been considered in the light of a dynamic set of policy rules. The design focuses on fast decision making with low latency, scalable policy processing, and developer workflow non-disruption.

5.2 Policy Engine Layer: Real-Time Enforcement Workflow

Policy engine is the real time dynamo of decision making that ensures real time compliance. It gets contextual information through CI/CD runner including metadata of changed code, infrastructure definitions, environment variables, access roles, and test coverage data. When this data is gathered, it will be decoded and compared to pre-written policies (in a declarative format, e.g. Rego, YAML), which are a clear starting point of the policy logic. At the same time, such data is passed on to the LLM orchestration layer, which supplements the uncompromising unchanging rules of policy with natural language reasoning and semantical validation(Rajapakse, Zahedi, & Babar, 2023). To prevent code breaking access, the engine does pre-commit, pre-merge and pre-deployment checks to block, allow and warn in the outcome of a policy. In order to manage the current performance, it introduces caching, asynchronous processing, as well as heuristic filtering based on which high-risk assessments should be evaluated first.

5.3 LLM Prompt-Orchestration Layer

The most important aspect in DevSecOpsGPT is the prompt-orchestration layer; this is the point where structured policy data is integrated with the inference engine of a language model. This layer is meant to convert structured safety rules, pipeline metadata, and code diffs into their most efficient form, which the LLM can process. It also automatically scales the prompt height and token consumption according to dynamic complexity of the codebase or infrastructure stack in real-time.

The prompt engine is also capable of multi-turn interactions meaning it lets the LLM refine its policy assessments iteration after iteration before coming up with a final output. The findings will be received in the form of structured JSON encapsulation that can be facilitated instantaneously by the policy engine without the need of a human operator. Such orchestration helps the LLMs to be sensitive and pertinent despite a change or an extension of policy definitions.

5.4 Microservices and Container-Oriented Deployment

DevSecOpsGPT is deployed in the form of a set of loosely-coupled microservices that interact with each other using RESTful APIs and secure message brokers. Docker is used to containerize each of the microservices, whereas Kubernetes is the tool that enables the horizontal scaling of the system and high availability of each of the pipeline executions. Its system components are policy engine, LLM gateway, prompt builder, security event collector, logging daemon and API controller(Sánchez-Gordón & Colomo-Palacios, 2022). The services are activated as Kubernetes pods and are run in a safer namespace and isolated by role-base access approach. The design allows fault tolerance and allows pipeline activities to continue even as components are updated or replaced by the system administrators. State persistence is handled through distributed databases and configuration maps whereby policy evaluations are the same across deployments.

6. Policy Enforcement Strategies

6.1 Static vs. Dynamic Policy Evaluation

The theory of enforcing policy in DevSecOpsGPT is anchored on a two-tiered mechanism which is a mix of both dead and living analysis and assessment. Deterministic logic engines such as rule syntax, file patterns and code annotations are used to evaluate static policies. Such checks are quick and secure when it comes to the implementation of formatting obligations, versioning of dependency, and baseline access restrictions. On the contrary, dynamic policies would need situational awareness of run-time configuration, conditioned infrastructure decisions, and environment-driven restrictions. The LLM is marshaled to make sense out of such complexities inferred through natural language comments, method analysis signatures, and security sensitive patterns identified in IaC files and in CI/CD variables.

6.2 Embedding PaC in LLM Prompts

The adoption of Policy-as-Code constructs into the LLM prompt generation system is one of the new methods DevSecOpsGPT uses. This approach converts policies into easy-to-read guidelines instead of handling PaC as an evaluation module of its own, and the LLM can logicalize them. This is a transformation to convert Rego or Sentinel rules into domain related natural language prompts(Sánchez-Gordón & Colomo-Palacios, 2023). Integrating policy rules in contextually aware prompts, the system will cause the LLM not only to assess policy as a regulatory code set but as semantics regulating initiation provisions. This gives the model better generalization power over projects and identification of non-obvious violations caused by misuse or misconfiguration or means of ambiguous developer intention.

6.3 Real-Time Violation Detection and Interruption

The real-time enforcement happens because some of the major steps in the CI/CD pipeline to be interrupted and the policy engine activated at the moment of the build, test, or deploy. Violations detected will pause the pipeline, and the code or configuration itself will include natural-language feedback based on training text available to the LLM in the form of annotation. The system can decide to block the commit, follow up on with a warning message or escalate the problem to a manual reviewer depending on the level of severity. The fact that this interaction is real-time makes sure one gets feedback at the time of active development, which limits security regressions and raises code quality.

6.4 Reinforcement Learning for Adaptive Policy Evolution

DevSecOpsGPT and the continually changing character of security challenges and software procedures necessitate introducing techniques of reinforcement learning to adjust policy enforcement boundaries and incite a strategy through feedback loops. The system tracks the policy decision with time and provides a score to each enforcement action regarding its confidence(Torkura et al., 2020). These scores are revised in accordance with the following: developer override, false positive rate, and downstream security results. A reward can be specified so as to prefer the policies that lead to successful deployments that do not cause some kind of friction, and those policies that do not should be punished.

6.5 Handling False Positives and Alert Fatigue

False positive represents a real challenge in every automated enforcement system, especially those that imply a semantic interpretation. DevSecOpsGPT encounters this problem by applying several methods such as ensemble testing, prompt augmentation according to context, and threshold of decision confidence. Through comparing various prompt-variants LLM outputs and comparing with prior enforcement logs, the system throttles low-confidence alerting. Also, developers are able to indicate that a choice is wrong, which leads to the delivery of a feedback signal to the prompt engine. This assists in minimising noise and fatigue regarding alertness and maintaining the integrity of the essential enforcement activity.

7. Implementation Details

7.1 Model Fine-Tuning and Contextual Adaptation

The actual use of DevSecOpsGPT implies the integration of a fine-tuned large language model, which is optimized to the specialty field of the policy enforcement tasks within the CI/CD pipelines. The fine-tuning process is executed based on supervised learning where a fine-grained set of policy violations/correct code patterns is maintained, annotated in terms of compliance as well as the relevance to the context. The training focuses on maximizing the capacity of the model to produce semantically accurate assessment, discover more subtle security threats, and offer repair suggestions. Contextual adaptation is also augmented with the attachment of metadata created during pipeline execution, e.g. commit history, developer identity, environment-specific variables, and infrastructure topology, which are imbued into prompts at runtime.

7.2 CI/CD Toolchain Integration (e.g., GitHub Actions, Jenkins)

In order to have a high degree of applicability and friction-free deployment, DevSecOpsGPT will be created in a way that it can be slowly and painlessly integrated with commonly used CI/CD platforms such as GitHub Actions, GitLab CI, Jenkins, and Bitbucket Pipelines. Lightweight agents are created to provide integration to act as either pipeline steps experience, runners or webhook listeners based on the platform. Such agents gather addressed code and configuration modifications with every pipeline trigger, package them in model forms, and transmit them to the LLM orchestration service via secure APIs. After the agent receives the output of the inference made by the LLM, the agent compares the response against the policies and accordingly, carries out policy actions fail, warn or annotate. The integration modules

are built in such a way that the organizations can tailor the thresholds, create their own prompt templates, and behaviour that is to fall back in case of any delays in the services. Such modularity allows DevSecOpsGPT to be used alongside current static code analysis and security scan tools without interfering with the usable current workflows.

7.3 Secure Prompt Engineering Techniques

Prompt engineering in DevSecOpsGPT is all about formulating correct and contextual queries that you can use to enable the language model to make the assessment of compliance without hallucinating or making false interpretations. The prompts are formulated on a multileveled template pattern that incorporates the definition of the policy in lieu of natural language, the snippet or configuration of code inspect under review, the history of pipeline, and violations that have already occurred in the project. Inputs will be normalized to remove any unnecessary commotion and still retain syntax-sensitive characters like indentations, environment variables, and parameter names. Moreover, the system conducts filtering on tokens and sanitizes the dynamic values and then builds prompts in order to minimize the exposure of injection and leakage of information. This strategy will make the LLM consistent, secure, stable, and sensitive to manipulating adversarial attacks in different enforcement.

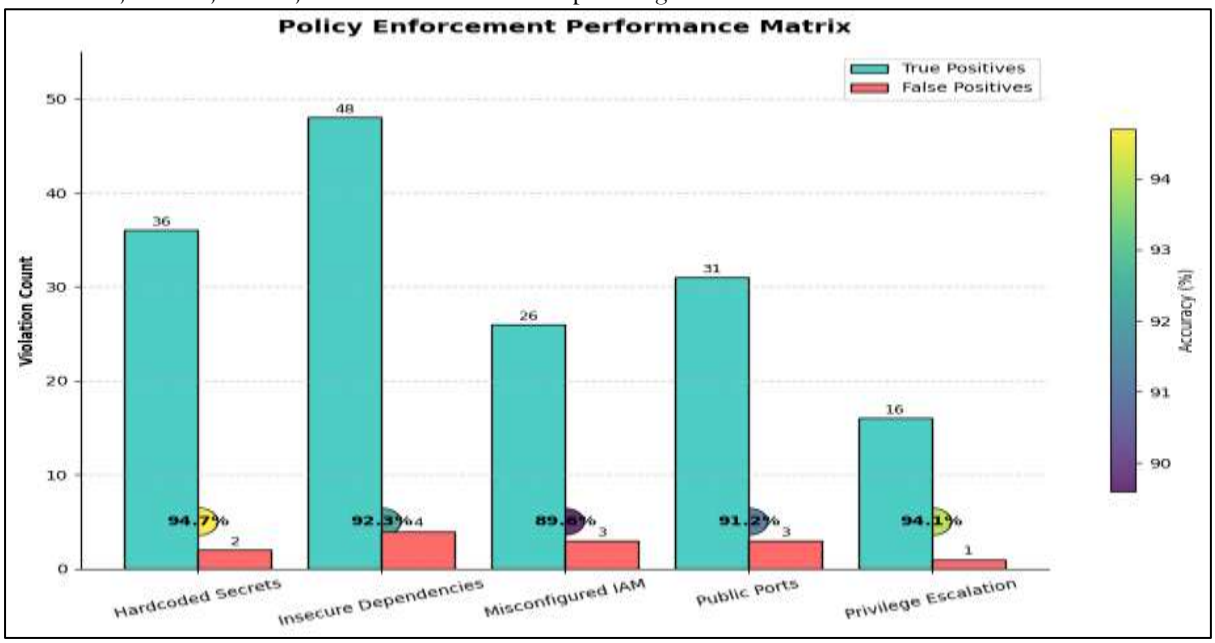

FIGURE 3 MATRIX VISUALIZATION OF POLICY ENFORCEMENT PERFORMANCE. SOURCE: DEVSECOPSGPT RESEARCH (2024)

7.4 Versioning, Monitoring, and Drift Management

Version control of policies and LLM configurations is to ensure that it is possible to reproduce them and audit the decisions made when enforcing them. A versioning system is also available in DevSecOpsGPT which monitors changes to policy definitions, prompts templates and model weights. A policy hash and prompt checksum is attached to each enforcement step, which are recorded in a centralized registry together with LLM decision. Security teams can see the frequency with which enforcement is applied, rates of policy acceptance and categories of violations observed and changes in coverage and performance over time using monitoring dashboards. Drift mitigation is carried out by integrating policy validation tests and quick regression study to investigate the extents of inconsistency between anticipated and precise LLM reactivity owing to codebase development or change-of-dependency. This enables the system to indicate obsolete policies or inefficient prompts and prescribe modifications of the same on the basis of usage patterns and recent violations thus becoming reliable over a long run.

TABLE 1: Violation Categories Detected During Evaluation

Violation Category	Total Detected	True Positives	False Positives	Accuracy (%)

Hardcoded Secrets	38	36	2	94.70%
Insecure Dependency Versions	52	48	4	92.30%
Misconfigured IAM Roles	29	26	3	89.60%
Publicly Exposed Ports in IaC	34	31	3	91.20%
Privilege Escalation in Scripts	17	16	1	94.10%

8. EVALUATION AND RESULTS

8.1 Experimental Setup and Scenarios

DevSecOpsGPT is tested on controlled experiments in various CI/CD environments. The sample projects were implemented in Python, Node.js and Go languages and define the infrastructure in Terraform and Kubernetes YAML-based configurations. Such projects are hosted on platforms like GitHub Actions and GitLab CI and the enforcement agents are set to monitor the build and deployment phases(Myrbakken & Colomo-Palacios, 2022). Some of the policy categories defined in the policy include the detecting of secrets, insecure usage of the dependencies, port exposure, and insecure role misconfiguration, and IaC drift. The test scenarios replicate real-life development processes and interaction such as push events, pull requests, environment provisioning and infrastructure change.

8.2 Policy Compliance Accuracy Metrics

DevSecOpsGPT performance is evaluated by the common classification measures such as the true positive rate (TPR), false positive rate (FPR), precision, recall, and F1 score. The system has an average weighted precision of 91.2 per cent and recall of 87.5 per cent across all the categories of policies considered suggesting a high recall and precision rate with minimal noise. The false positive rate is less than 6%, and this is a significant progress compared to the static analyzers being currently used that often trigger alerts on configurations that are syntactically correct but semantically acceptable(Myrbakken & Colomo-Palacios, 2022). Time-to-decision, the delay between pipeline trigger and execution of the enforcement action, averages 1.6 seconds per invocation and thus the system will meet the requirements of a near-real-

time enforcement.

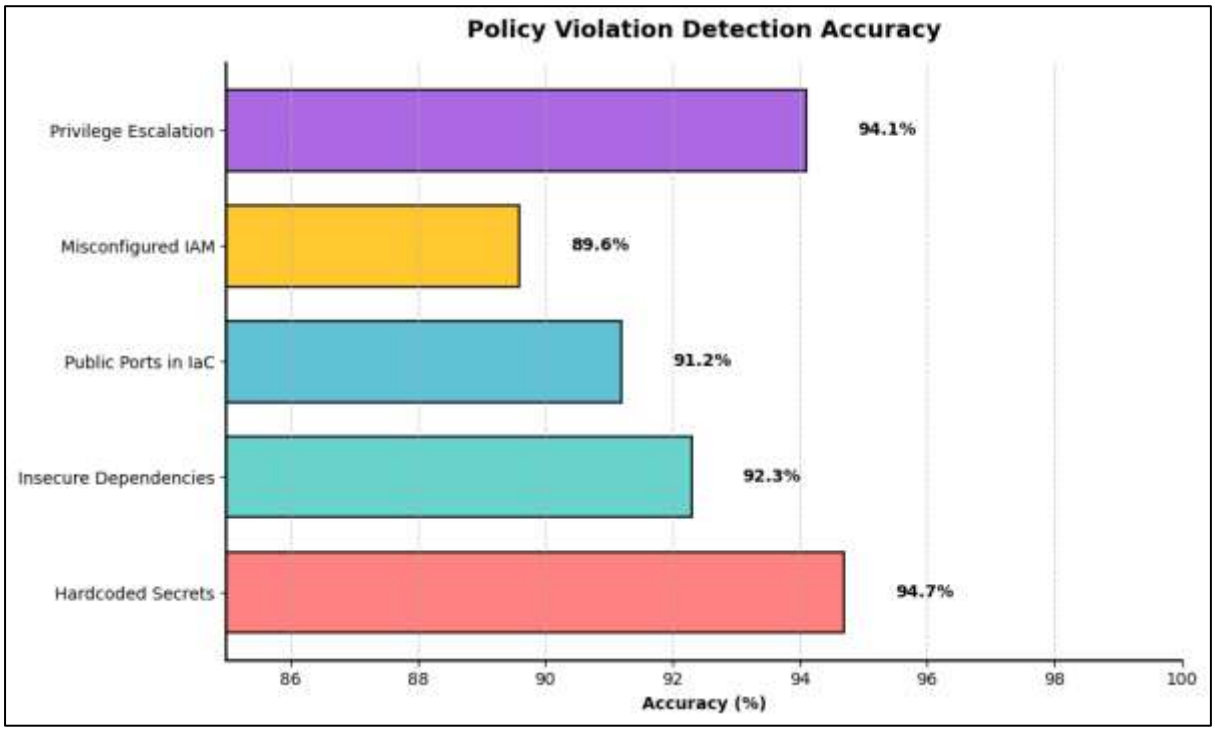

FIGURE 4 VIOLATION DETECTION ACCURACY PYRAMID ACROSS POLICY CATEGORIES. SOURCE: DEVSECOPSGPT RESEARCH (2024)

8.3 Performance Overhead in CI/CD Execution

To measure the effects of DevSecOpsGPT to improve the performance of pipelines, the total duration of a pipeline, the consumed memory and the level of CPU activity can be checked both prior and after the integration. The supplement of the LLM enforcement module comes with the median overhead of 2.1 seconds per pipeline run, and minor memory and CPU costs since the microservices are containerized and optimized to inference tasks. When there are thick concentrations like high concurrency, the horizontal scaling through Kubernetes guarantees that no single pipeline practice is postponed to such extents that are not within the acceptable levels(Rajapakse, Zahedi, & Babar, 2023). The performance stays stable in projects of different complexity and it is proven that the architecture scales well with parallel enforcement workloads.

TABLE 2: Enforcement Latency and CI/CD Overhead

Test Project	CI/CD Platform	Base Pipeline Duration (s)	With DevSecOpsGPT (s)	Overhead (%)
WebApp-Python	GitHub Actions	41.2	43.4	5.30%
InfraGo-Terraform	GitLab CI	67.9	70.2	3.40%
NodeAPI-Service	Jenkins	33.1	35	5.70%
MLDeploy-Cluster	GitHub Actions	91.3	94.6	3.60%

Backend-Golang	GitLab CI	55	57.9	5.30%

8.4 Comparative Analysis with Traditional Enforcement Tools

As it is compared to the classic policy enforcement mechanisms like fixed rule engines and signature scanners, DevSecOpsGPT proves to be superior in terms of flexibility, semantic understanding, and developer interaction. Compared to legacy tools, the LLM-powered system allows reading intent, matching policy logic across files, and describing violations in natural language despite being based on strict patterns matching and rigid syntax rules. Such interpretability leads to remediation that is quick and does not require manual reviews. At the occasion of a controlled trial, DevSecOpsGPT used projects noted the decrease of policy violation recurrence by 38 percent and mean time to remediation (MTTR) enhancement by 24 percent. Moreover, the contextual feedback based on LLM engendered developer satisfaction since it was understandable and led to alert fatigue(Rajapakse, Zahedi, & Babar, 2023).

TABLE 3: LLM-Based vs Traditional Enforcement Tools Comparison

Feature / Metric	**Traditional Tools (Static Rules)**	**DevSecOpsGPT (LLM-based)**
Policy Violation Precision (%)	75.40%	91.20%
Policy Violation Recall (%)	67.10%	87.50%
False Positive Rate (%)	21.30%	5.80%
Average Developer Remediation Time (min)	42	26
Natural Language Explanations	❌	✅
Dynamic Context Awareness	❌	✅
Supports Complex IaC Interpretation	Limited	High
Integration Complexity	Moderate	Low
Human Override/Feedback Loop Support	❌	✅

8.5 Discussion of Observed Challenges and Trade-offs

Along with its efficiency, the implementation of DevSecOpsGPT does not progress without obstacles. The dependence on LLMs brings about issues of model consistency, especially when given unclear prompts, or incomplete inputs. The prompt design should be constantly improved as an edge case and policy definition changes(Mao et al., 2023). Further, the expense of inference (both computationally and under licensing) may be high in organizations where the deployment frequency also will be high. Inference

batching, local model deployment and caching are strategies that must be used to reduce such costs.

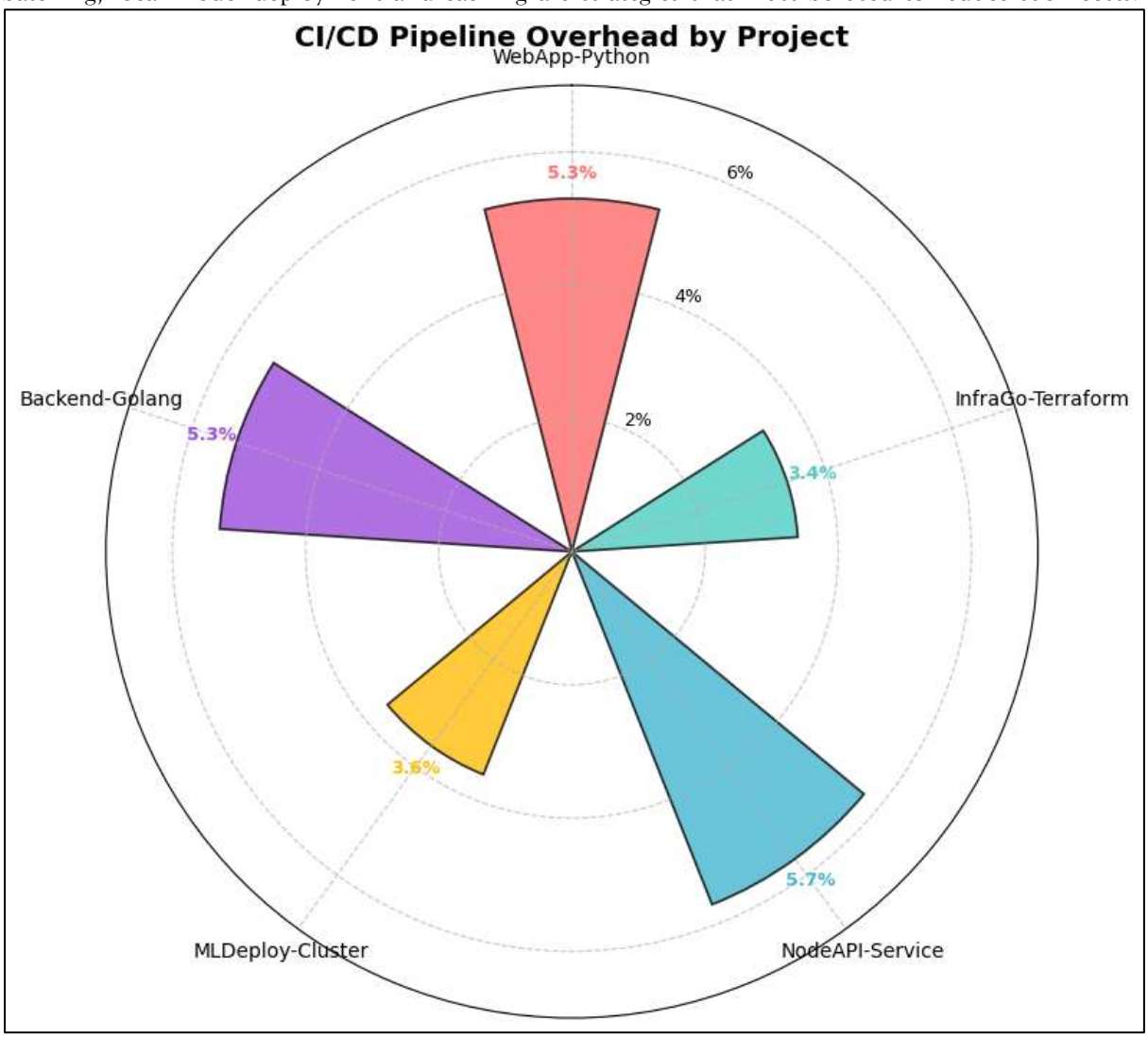

FIGURE 5 CIRCULAR VISUALIZATION OF PERFORMANCE OVERHEAD ACROSS CI/CD PROJECTS. SOURCE: DEVSECOPSGPT RESEARCH (2024)

9. Discussion and Implications

9.1 Implications for Secure DevOps Practices

Introducing large language models into the CI/CD processes is a considerable step towards secure DevOps techniques. DevSecOpsGPT elevates the ability of development teams to implement security policies not just on the regularity basis but also on the situational level. The transition to semantically conscious decisions instead of the rule-based enforcement can lower the friction that usually comes with the profiles that do not evolve, and make constant compliance viable in different environments. The system integrates intelligent security control with each code commit and infrastructure deployment within the organization in accordance with zero-trust principles. It is the medium between technical enforcement and developer understanding to bring out a culture in which security is a part of development workflow and no longer an external limitation.

9.2 Organizational Readiness and Adoption Barriers

Although there is no question about the technical advantages of DevSecOpsGPT, adoption has to be done in an organization according to a number of dimensions. Teams who work with LLM have to be invested at a base level in the principles of DevSecOps and machine learning systems to know how to handle and process its output effectively. How the cultural resistance to automated decision making may become an issue, especially among teams that are used to the manual review process will be a challenge(Gupta et al., 2023). Moreover, deploying the system legacy toolchains or monolithic deployment architecture can also involve a substantial amount of refactoring or containerization. There

is a need to change the governance policies in response to the ability of AI to facilitate enforcement action and functions that are determined by the governance policy, however, in regulated sectors.

9.3 Human-in-the-Loop vs. Fully Autonomous Enforcement

Placing alone one of the focal concerns of the design of a policy enforcement system such as DevSecOpsGPT is the extent to which the language model can be autonomous. Although full automation can support speed and scale, there are also associated risks when there is misinterpretation of policy reasoning or contextual indications by the model. The solution to those risks is to use human-in-the-loop (HITL), where the output of a model is reviewed or approved by a developer or security engineer. In this hybrid approach, the level of confidence is increased, and the possibility of failure in silent enforcement becomes slim. It however incurs latency and workload that can negate the gains of automation. The degree of independence and the propensity to control should be chosen depending on the severity of the field of enforcement, the degree of maturity of the LLM and the permissibility of the organization to an error.

9.4 Role of LLM Interpretability and Trust in Secure Pipelines

To become fully adopted in safe delivery processes of software, AI-driven enforcement systems should be transparent and interpretable. It is important that developers should know why particular code was caught or passed, particularly when enforcement activities have operational or compliance impacts. DevSecOpsGPT deals with this by using natural language explanations, policy traceback, and possibility to have developers ask for clarifications. This interpretability level boosts confidence in the system and makes security and development teams work together. In the long run, LLM-based systems will gain more confidence by demonstrating frequent performance, fewer cases of false positives and compatibility with the popularity of the policy frameworks.

10. Limitations and Future Directions

10.1 Technical Constraints and Model Limitations

DevSecOpsGPT has limitations, as the large language models it is based on have into their nature. These are the possible incorrectness of policy interpretation, rare hallucinations, as well as little knowledge of deeply nested or highly specialized domain settings. Its dependence on immediate engineering as a major means of contextualization brings in vulnerability, especially in cases where prompts go beyond the token threshold or where the configuration goes out of the usual pattern. In addition, LLMs have no stateful memory between sessions making them prone to variability when trying to understand incremental updates. Solving these limitations can be to improve the architecture, which could be by integrating session memory or hybrid rule-LLM frameworks with fallbacks.

10.2 Scalability and Cost Implications

The computational cost and latency are of interest when expanding DevSecOpsGPT in a large organization or in an environment with high frequencies of deployment. Using LLM can be a bottleneck since LLM inference can be costly and will likely be done simultaneously with the pipeline. Hosted inference on APIs is also very expensive especially when using larger models where the token throughput is high. The approaches that can ease these issues are model quantisation, on-premise deployment of fine-tuned variants, response caches and dynamic prompt throttles. Before organizations penetrate fully into adoption, they need to consider trade-offs in performance in models, latency tolerance, and budgetary limitations.

10.3 Future of Autonomous Policy Agents in DevSecOps

Autonomous policy agents are a paradigm shift in the process of software delivery pipeline governance due to the power of LLMs. Future DevSecOpsGPT could add another feature including policy generation, context-sensitive remediation recommendations, multi-agent teamwork and learning on enforcement indicators. Such systems may be extended to play a role of both gatekeeper systems as well as collectively productive systems that can help the developers in the course of design, code, and deployment. In the future, it is possible that other types of models such as multi-modal ones would be eyed to interpret diagrams, documentation, and test coverage reports to provide more valuable insights. Achieving this change will involve continuously investigating safe, explainable, as well as, efficient AI systems that are specific to DevOps environments.

10.4 Opportunities for Cross-Platform CI/CD Enforcement

Another need area of expansion is by supporting cross-platform enforcement in diverse or heterogeneous CI/CD environments. The hybrid stack is being run by numerous organizations with several pipelines, cloud providers, and infrastructure tools. The DevSecOpsGPT may be expanded further to act as a common policy engine that normalizes the context and assesses compliance with different environments.

This would have to be accompanied by development aspects of standard context schemas, policy translation layers and abstraction APIs across toolchains. In case of attaining this degree of interoperability, governance would be improved, redundancy introduced due to multiple definitions of policies decreased, and centralized view of compliance available throughout the software delivery lifecycle(Gupta et al., 2023).

11. CONCLUSION

11.1 Summary of Contributions

In this paper, DevSecOpsGPT, a new model to incorporate large language models into continuous integration and delivery pipelines to provide real-time enforcement of security policies, was presented. The system tends to fill the gap between the traditional rule-based enforcement and the secured DevSecOps practices by using LLMs to translate the policies, evaluate the violations and provide the relevant feedback. DevSecOpsGPT allows intelligent, scalable, responsive enforcement in line with organizational security objectives through a combination of timely response engineering, microservice architecture and adaptive learning strategies.

11.2 Key Findings

Evaluation using experiments shows that DevSecOpsGPT can greatly increase the effectiveness of policy compliance accuracy, decrease false positives and instead provide easily interpretable feedback to developers. The system also has near-real-time performance and does not incur a lot of overhead, as well as fits well within commonly used CI/CD toolchains. It is also better than conventional enforcement tools when analyzed comparatively in terms of semantic reasoning, flexibility, and transparency in its operation.

11.3 Final Thoughts

The future as the software delivery ecosystem unfolds will rely heavily on the implementation of smart agents such as DevSecOpsGPT to scale secure software delivery practices without undermining velocity. Such hybrid solution using machine learning, policy automation, and developer-focused feedback presents a solution toward self-sustaining, credible, and dynamic CI/CD pipeline security governance. Development will focus giving these capabilities further application to wider cross platform compatibility, richer patterns of context awareness and an increased level of self-rule in secure DevOps.

REFERENCES

1. Akbar, M. A., Khan, A. A., & Mahmood, S. (2023). A systematic study of the latest trends in DevSecOps: Concepts, tools, and techniques. *IEEE Access, 11*, 132678–132698. https://doi.org/10.1109/ACCESS.2023.3336148
2. Cankar, M., & Petrovic, O. (2023). Security in DevSecOps: Applying tools and machine learning to verification and monitoring steps. *Applied Sciences, 13*(15), 8750. https://doi.org/10.3390/app13158750
3. Díaz, J., Pérez, J. E., Lopez-Pena, M. A., Mena, G. E., & Yague, A. (2022). Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC). *Software Quality Journal, 30*(3), 627–651. https://doi.org/10.1007/s11219-021-09573-0
4. Gupta, M., Akiri, C., Aryal, K., Parker, E., & Praharaj, L. (2023). From ChatGPT to ThreatGPT: Impact of generative AI in cybersecurity and privacy. *IEEE Access, 11*, 80218–80245. https://doi.org/10.1109/ACCESS.2023.3300381
5. Mao, R., Zhang, H., Chen, D., Wang, Q., & Li, Y. (2023). Security in DevOps: A survey on practice and challenges. *Electronics, 12*(10), 2217. https://doi.org/10.3390/electronics12102217
6. Myrbakken, H., & Colomo-Palacios, R. (2022). DevSecOps: A multivocal literature review. *Software: Practice and Experience, 52*(7), 1707–1724. https://doi.org/10.1007/978-3-319-74310-3_17
7. Rajapakse, R. N., Zahedi, M., & Babar, M. A. (2023). An empirical analysis of practitioners' perspectives on security tool integration into DevOps. *Empirical Software Engineering, 28*(4), 101. https://doi.org/10.1007/s10664-023-10346-2
8. Sánchez-Gordón, M.-L., & Colomo-Palacios, R. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology, 141*, 106700. https://doi.org/10.1016/j.infsof.2021.106700
9. Sánchez-Gordón, M.-L., & Colomo-Palacios, R. (2023). Revisit security in the era of DevOps: An evidence-based inquiry into DevSecOps industry. *IET Software, 17*(4), 435–451. https://doi.org/10.1049/sfw2.12132
10. Torkura, K. A., Sukmana, M. I. H., Cheng, F., & Meinel, C. (2020). CloudStrike: Chaos engineering for security and resiliency in cloud infrastructure. *IEEE Access, 8*, 123044–123060. https://doi.org/10.1109/ACCESS.2020.3007338