NEXUS: A Policy-Driven Governance Layer for Authority Transfer in Self-Healing CI/CD Pipelines

A Declarative, Auditable Boundary Between Autonomous Fix Generation and Human Authority, Deployed as a Cloud-Hosted Service

[1]Anjali Shukla, [2]Trupti Asolkar
[1]M.Sc. Computer Science, [2]Asst. Professor
Department of Advanced Computing
[1], [2]Nagindas Khandwala College,
[1], [2]University of Mumbai, Maharashtra, India

Abstract: Self-healing CI/CD tools now. Patch broken pipelines automatically, but almost all of them apply the same fix-generation process to every file in a changeset, whether it is a test or a workflow file governing deployment approval. A 2026 survey by Barnes, Ghaleb and Hassan names this the authority transfer" problem: how much control an organization hands to an autonomous agent. This paper presents NEXUS, a governance layer that classifies every file in a failing changeset as either data-plane (to auto-fix) or control-plane (must involve a human) using a declarative policy file. If any file is control-plane, the whole changeset is blocked from auto-merge and routed to a human Review Queue; only fully data-plane changesets are auto-proposed. NEXUS integrates with GitHub and GitLab repositories generates fixes via a low-latency cloud-hosted LLM backend (Groq-served Llama models), runs as a continuously available service on AWS EC2 and maintains a complete audit log. This paper presents its architecture, classification logic and a literature-grounded comparison against existing approaches, none of which implement an authority boundary of this kind.

Keywords: Continuous Integration/Continuous Deployment (CI/CD) Authority Transfer, AI Governance, Policy-as-Code, Data-Plane, Control-Plane, Self-Healing Pipelines, Human-in-the-Loop, Large Language Models, Cloud Deployment, DevSecOps.

1. INTRODUCTION

Continuous Integration and Continuous Deployment (CI/CD) pipelines are now the backbone of software delivery, automating build, test and deployment steps in corporate and enterprise environments. As pipelines grow more complex, failures happen often. The manual work of finding a build, locating the file that caused it and committing a fix slows down developer productivity.

Recent self- healing CI/CD tools tackle this problem by using AI to read failure logs and automatically propose a fix. This is already real: LogSage has been tested on large real- world pipeline data. Shows high precision and Repair Agent goes further by treating the language model as an agent that can investigate a failure with tools instead of creating a single patch [2] [3]. Decades of research on self- healing and self- adaptive systems [4] [5] provide the background for these tools. None of the fix- generation tools discussed here can differentiate the type of file they are editing. An AI agent that can edit a failing test file can also edit the workflow YAML that decides who must approve a production deployment or a file that holds access policy. A 2026 survey by Barnes, Ghaleb and Hassan calls this the authority transfer problem. Says that there is still no standard boundary for it [1].

This paper introduces NEXUS, a system that closes that gap. Then proposing a new fix- generation model NEXUS inserts a governance step before fix generation. NEXUS first classifies every changed file using an auditable policy; only changesets that are entirely safe are allowed to auto- merge. All other changesets are given to a human. NEXUS is built as a working system. Runs continuously in the cloud not just as a local prototype. The rest of this paper reviews related work describes the problem and presents NEXUS architecture, implementation and evaluation.

2. LITERATURE REVIEW

2.1 Existing Tools and Methodologies

Two systems are presented in the literature, which demonstrate that automated CI/CD fix generation also works in settings and point to where NEXUS can

improve. For example, LogSage works in production and has been benchmarked on numerous valid runs of real pipelines with high precision. LogSage fixes all files in a changeset, regardless of file risk [2]. RepairAgent uses the language model as an agent to investigate a failure using tools such as the reading of logs and inspection of files. But RepairAgent can give better fixes, but it still suffers the same lack of intelligence: RepairAgent does not know which files it should repair [3].

2.2 Related Research Paper

The problem framing for this project is based on the paper 'Human-centred design for the creation of E-Tools' written by Barnes, Ghaleb and Hassan in 2026. It suggests that 'authority transfer' is the level of control an organization grants to an autonomous agent and that there is no universal answer to this question [1]. A series of related studies backs up design decisions of NEXUS, but they are not specific to CI/CD. Studies on trust calibration and human AI collaboration show how to build trust in human AI interaction [6] how decision support, autonomous systems and LLM agents can converge [7] a model for transparent and accountable AI agents [8] and methods for trust, risk and security management in agentic multi agent systems [9]. It is these works that have casted doubt on the use of model confidence as the sole measure for high risk changes without the involvement of a human. Research into policy as code is ongoing: ARPaCCino's retrieval augmented approach to policy as code compliance checking [10] and another paper looks at reusing policy as code across the CI/CD and Kubernetes admission control [11]. The results of both studies justify using a policy file, not a learned classifier. Last but not least, research into AI for DevSecOps [12] devs adoption of Terraform security policies [13] quality metrics [14] and deployability [15] focus on infrastructure security that NEXUS's control plane category aims to safeguard

3. PROBLEM STATEMENT

Current self-healing CI/CD tools run the automated

fix generation process on every file in a failing changeset no how big the file's impact. A broken unit test and a workflow file that removes a required approval gate look the same to the fix-generator. Both become just 'a file that needs to change to make the pipeline pass.' This poses a risk: an AI agent tasked with 'fixing' a pipeline blocked by a required-reviewer rule could meet that goal by editing the workflow YAML to delete the rule thereby removing an approval gate instead of fixing the real problem. Even though the authority-transfer gap is named in the literature [1] none of the tools discussed above has a boundary that decides before a fix is generated or merged whether an AI agent is allowed to touch a particular file. NEXUS fills this gap.

4. OBJECTIVES OF THE RESEARCH

- To design a declarative classification policy that separates data-plane files (application and test code safe for auto-fix) from control-plane files (workflow configuration, secrets, approval gates, access policy) based on the authority-transfer gap found in literature.
- To implement a verdict rule. If any file in a changeset is control-plane the whole changeset is blocked from auto-merge than allowing a partial auto-merge of a mixed changeset.
- To add a risk score on top of the binary verdict only to help a human reviewer prioritize escalated items without affecting the auto-merge decision.
- To build fix-generation using a large language model backend served in production by a low-latency cloud-hosted inference provider. The system is restricted so that the model runs on changesets that have already been cleared as data-plane or that have received explicit human approval.
- To create a human Review Queue, the sole route through which a control-plane-flagged fix can be approved and released
- To create a human Review Queue, the sole route through which a control-plane-flagged fix can be approved and released
- To keep an audit log that records every classification decision for both automated and interactive flows.
- To enable integration with GitHub and GitLab repositories allowing operators to choose a repository rather than using a simulated or offline dataset
- To deploy the system as a running publicly accessible cloud service instead of a machine-local prototype.

5. PROPOSEDMETHODOLOGY AND SYSTEM ARCHITECTURE

5.1 Overall Methodology

NEXUS uses an event-driven approach. When a CI/CD issue occurs the system can detect it automatically via a GitHub webhook. When a user

connects a repository manually. The issue triggers a fixed series of steps. The changed files from the commit or pull request are retrieved from the provider's API. Each file is examined to determine whether it belongs to the data-plane or control-plane using a set of rules. A risk score is then calculated. A decision is made for the set of changes. If any file is classified as control-plane the decision is to send the set to a human for review. When all files are data-plane does the system automatically merge the changes. The event is stored in a storage system, which serves as the audit log. If the changes qualify for merging, a system that uses a cloud-hosted language model generates a patch and opens it as a draft pull or merge request. If the changes require review a fix may still be created. The patch is kept in a queue until a person approves or rejects it. This process—checking, deciding, acting or forwarding to a human and recording—repeats for every problem that NEXUS identifies. It remains active in its cloud setup continuously not just when needed.

5.2 System Architecture

System Architecture The structure of NEXUS is divided into six parts. Each part communicates with the others through an interface. This design allows components such as the event store or the language model backend to be replaced without affecting the rest of the system. The entire system how changes are taken in classified, fixed escalated and viewed runs as a service on an AWS EC2 instance. It is not composed of processes.

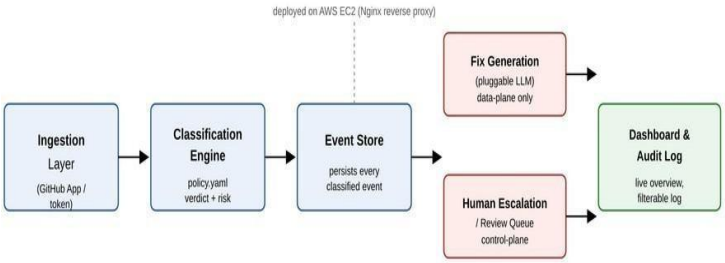

Figure 5.2: NEXUS System Architecture

5.3 System Components

Table 1 presents the components of the proposed NEXUS system

Components	Description
Classification Engine	Loads policy.yaml, matches each changed file against glob rules and returns a verdict for the entire changeset along with a deterministic risk score.
Ingestion Layer	Detects a failure GitHub App + webhook) or interactively (connected GitHub/GitLab token via MCP/REST); also lists every accessible repository so monitoring can be scoped to one repository instead of only the automatic feed.
Event Store	Persists every event, supports the audit log, and can be swapped for a real database.
Fix Generation Engine	An LLM backend served in production via Groq's low-latency hosted Llama inference (with a local-model fallback for offline development) creates a fix limited to data-plane or human-approved files and filtered by log relevance.
Human Escalation / Review Queue	For any control-plane flagged changeset, opens a GitHub Issue or a Review Queue item that requires human approval or rejection.
Dashboard & Audit Log	Overview, searchable and filterable audit log and an in-browser policy YAML editor accessible over HTTPS from the AWS EC2 deployment.

Table 1: System Components

5.4 Workflow Diagram

The diagram shows how a failing set of changes moves through NEXUS. A problem is found the files are checked against the rules. A decision is made. If all files are data-plane the system creates a fix. Opens a draft pull or merge request. If any file is control-plane the set of changes enters the Review Queue where a person must approve or reject it before any changes are made in the repository. Every step checking making a fix getting approval or rejectionis

logged in time

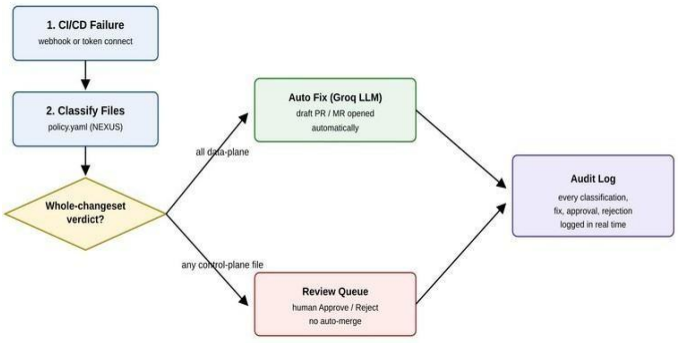

Figure 5.4: NEXUS Workflow

6. IMPLEMENTATION DETAILS

NEXUS is made up of two parts: a Node.js/Express server and a React/Vite website. It is built around the six parts described above and runs as a service.

• Classification: The rules in policy.yaml are turned into matchers when the system starts. The classification checks the "any control-plane file blocks everything" rule. Gives each file a plane the rule it matched and a risk score from 0 to 1 (secrets/IAM ≈ 1.0 Dockerfiles ≈ 0.91 workflow files ≈ 0.79, data-plane ≈ 0.15). This helps a person decide what to do. The decision is not changed by the score. A file that matches none of the rules is considered control-plane by default.

• Ingestion: There are two ways to get changes – a path using GitHub App and a manual path using a personal access token. The manual path checks the live keeps it in memory and never saves it to disk. It talks to GitHub through a GitHub server and to GitLab through its REST API. A tool lists every repository the token can access so monitoring can be limited to an one instead of just using the automatic feed.

• Fix Generation: In production the fix is made using Groqs Llama model. This was chosen after testing with a model (Ollama) showed the time it took was too long for a review process. A local model is still available for use when there is no internet. Every fix prompt tells the model to keep parts of the file the same so changes are not made by accident. A filter looks atthe error log to find keywords and skips making a fix for files that are not related letting the user know it was skipped.

• Log excerpt extraction: It uses the line the CI system writes when a step fails, not the end of the log, which's often not relevant.

• Escalation: It creates a GitHub Issue that lists which files are control-plane and which rule they match. No code is. No model is used on this path.

• Event Store: It is a file today and four functions are used to access it so it can be replaced with a database like Postgres or SQLite without changing any code.

• A command-line tool can check files directly a git diff that is piped in or flags like --staged or --range and returns 0 (eligible for merge) or 1 (needs human review) so it can be used as a gate in a CI pipeline.

• Deployment: NEXUS is running as a service on an AWS EC2 (Ubuntu 22.04) instance. The backend and the website are served together behind an Nginx reverse proxy. Access is limited to SSH, HTTP and HTTPS only so the system is available as a service and not just as a local demo.

• Security: Uses CORS that only allows the origin checks the signature of webhooks and has secure cookies. Files that do not match any rule are considered control-plane by default

7. EXPERIMENTAL EVALUATION

7.1 Evaluation Setup

NEXUS adds a layer of control in front of fixing code, not a way to create fixes. There is no labeled set of data to test it against yet. Its evaluation in this paper is based on design and a comparison with tools. It compares with existing self-healing and policy-as-code approaches to see if they have an authority boundary. It also shows an example of how the system works with real changesets using the live system on AWS EC2, not a fake test environment.

7.2 Evaluation Metrics

The system is tested using these measures, chosen to show what an authority boundary needs to do of just how good the fixing is:

• Classification accuracy: Does the system give the decision that its rules say it should for a given set of changes?

• Escalation dependability: Does every set of changes that has a control-plane file get blocked from being merged?

• Traceability: Can every decision, fix, approval and rejection be found in the log?

• Comparison: Do existing self-healing and policy-as-

code tools have the kind of authority boundary?

- (Future work) How well does the system classify changes compared with a set of labeled fixes from the past looking at cases where a control-plane change was accidentally merged and cases where a safe change was wrongly marked as needing review compared to a system, without this boundary.

7.3 Comparative Analysis

Table 2 compares NEXUS against the two self-healing fix-generation tools reviewed in Section 2.1 and against ARPaCCino, the closest existing policy-as-code approach [10] along the dimensions the authority-transfer gap makes relevant.

Tool	Fix Generation	File-Risk / Authority Classification	Human-in-the-loop Boundary	Auditability
LogSage [2]	Yes, at scale	No. Uniform across all files	No	Partial
RepairAgent [3]	Yes, agentic/tool-using	No. Blind spot	No	Partial
ARPaCCino [10]	No. Compliance checking, not a fixer	Policy-as-code compliance, not a data/control-plane split	Not mandatory	Depends on RAG trace
NEXUS (deployed)	Yes, Groq-hosted LLM backend	Yes. Whole-changeset	Yes Yes. Review Queue	Full. Versioned policy + event log

Table 2: Comparative Analysis

7.4 Classification Scenarios

Because the classification engine is fully deterministic its behaviour can be demonstrated directly against Representative changesets using the real policy.yaml rule set described in Section 5.3–6 rather than a simulated test harness. Table 3 walks through four scenarios.

Scenario	Changed Files	Matched Policy Rule(s)	Verdict	Risk
A	tests/ test_auth.py , Dockerfile, src/utils.py	**/tests/**, **/Dockerfile, **/*.py (all data-plane)	AUTO_MERGE_ELIGIBLE	0.15
B	.github/ workflows/ ci.yml, tests/test_auth.py	**/.github/ workflows/ ** (control-plane) present alongside a data-plane file	ESCALATE_TO_HUMAN (whole changeset, incl. the safe test file)	≈ 0.79
C	config/ deploy-approval.yaml	**/deploy-approval* (control-plane)	ESCALATE_TO_HUMAN	≈ 0.95
D	infra/ notes.txt	No explicit rule matched. Conservative default applies	ESCALATE_TO_HUMAN	0.70 (defaulted)

Table 3: Classification Scenarios

7.5 Results and Observations

The comparison in Table 2 shows that the authority-transfer gap is not merely a claim in a survey paper [1]. It is a absent feature across the concrete tools examined including ARPaCCino, whose retrieval-augmented compliance checking is the closest conceptual neighbour to NEXUSs policy layer but is not built around a data-plane/control-plane authority split or a mandatory human-approval step for fix shipping [10]. The classification walkthrough in Table 3 confirms the policy engine behaves as specified: conservatively (Scenario D defaults to escalation rather than assuming safety) and at the level of the whole changeset rather than the individual file (Scenario B escalates a safe test file purely because it shares a changeset with a workflow file).

7.6 Discussion

NEXUSs declarative policy.yaml approach trades some of the adaptiveness of a learned policy for a boundary that can be read versioned and reasoned about line by line. Consistent with the case for policy-as-code over black-box automation made in [10] and [11]. The main limitation of the work is that it has not yet been validated against a large hand-labeled real-world dataset as proposed in Section 7.2; that empirical validation is the immediate next step discussed further in Section 10. An earlier limitation, encountered during development was fix latency and quality under the free local-model (Ollama) default; switching the production default to Groqs hosted Llama inference resolved this bottleneck and the local-model path is retained only as an offline fallback, not the deployed default.

7.7 Results and Observations

The comparison in Table 2 shows that the authority-transfer gap is not merely a claim in a survey paper [1]. It is a absent feature across the concrete tools examined including ARPaCCino, whose retrieval-augmented compliance checking is the closest conceptual neighbour to NEXUSs policy layer but is not built around a data-plane/control-plane authority split or a mandatory human-approval step for fix shipping [10]. The classification walkthrough in Table 3 confirms the policy engine behaves as specified: conservatively (Scenario D defaults to escalation rather than assuming safety) and at the level of the whole changeset rather than the individual file (Scenario B escalates a safe test file purely because it shares a changeset with a workflow file).

7.8 Discussion

NEXUSs declarative policy.yaml approach trades some of the adaptiveness of a learned policy for a boundary that can be read versioned and reasoned about line by line. Consistent with the case for policy-as-code over black-box automation made in [10] and [11]. The main limitation of the work is that it has not yet been validated against a large hand-labeled real-world dataset as proposed in Section 7.2; that empirical validation is the immediate next step discussed further in Section 10. An earlier limitation, encountered during development was fix latency and quality under the free local-model (Ollama) default; switching the production default to Groqs hosted Llama inference resolved this bottleneck and the local-model path is retained only as an offline fallback, not the deployed default.the binary auto-merge/escalate verdict fully rule-based and auditable while allowing a learned model to refine only the human-facing prioritisation score. Finally extending provider support beyond GitHub and GitLab and formalising policy.yaml as a shareable artefact across organisations. In the spirit of [11]. Would broaden the system's applicability.

11. REFERENCES

[1] Barnes, M. E., Ghaleb, T. A., & Hassan, S. (2026). Autonomy and control in CI/CD pipelines: Rethinking it from assistance to agency. https://doi.org/10.1145/3805760.3814897

[2] Xu, W., et al. (2025). LogSage: An LLM-based Framework for CI/CD Failure Detection and Remediation with Industrial Validation. https://doi.org/10.1109/ase63991.2025.00310

[3] Bouzenia, I., Devanbu, P., & Pradel, M. (2025). RepairAgent: Program Repairing Agent based on LLM. Proceedings of ICSE 2025. https://doi.org/10.1109/icse55347.2025.00157

[4] Psaier, H., & Dustdar, S. (2011). Self-healing system survey: approaches and systems. Computing, 91(1), 43-73. https://doi.org/10.1007/s00607-010-0107-y

[5] Angelis, A., & Kousiouris, G. (2025). Current landscape of self-adaptive cloud design and operations patterns: Goals, strategies, tools, evaluation and dataset views. https://doi.org/10.3390/fi17100434

[6] Mehrotra, S., et al. (2023). Systematic review: promoting trust for human-AI interaction. https://doi.org/10.1145/3696449

[7] Dong, A., et al. (2026). A systematic review and collaboration convergence framework for human-AI collaboration, including decision support and autonomous systems, and LLM agents. Sustainability, 18(11), 5313. https://doi.org/10.3390/su18115313

[8] Cheng, E. C., Cheng, J., & Siu, A. (2026). Toward safe and responsible AI agents: A three-pillar framework for transparency, accountability and trustworthiness. https://doi.org/10.48550/arxiv.2601.06223

[9] Raza, S., et al. (2025). TRiSM for agentic AI: A Review of Trust, Risk, and Security Management for LLM-based agentic multi-agent systems. https://doi.org/10.48550/arxiv.2601.06223

[10] Lupinacci, M., Blefari, F., Romeo, F., Pironti, F. A., & Furfaro, A. (2025). ARPaCCino: Agentic-RAG for Policy as Code Compliance. https://doi.org/10.1007/978-3-032-05727-3_39

[11] Nogueira, L. (2026). An empirical evaluation of governance consistency when applying policy-as-code in CI/CD and Kubernetes admission control. Computers (MDPI). https://doi.org/10.3390/computers15070453

[12] Fu, M., Pasuksmit, J., & Tantithamthavorn, C. (2025). The landscape of AI for DevSecOps and prospects. ACM Transactions on Software Engineering and Methodology, 34(4):115. https://doi.org/10.1145/3712190

[13] Verdet, A., Hamdaqa, M., et al. (2024). Analysis of developers' Terraform security policies by cloud provider. Empirical Software Engineering, 29, article 10610. https://doi.org/10.1007/s10664-024-10610-0

[14] Begoug, M., Chouchen, M., & Ouni, A. (2024). Terrametrics: A framework for infrastructure-as-code (IaC) quality metrics in Terraform that is open source. Proceedings of ICPC 2024. https://doi.org/10.1145/3643916.3644441

[15] Zhang, T., et al. (2025). Infrastructure as code generation with a focus on deployability: Fail, learn, refine and succeed through LLM-empowered DevOps simulation. https://doi.org/10.1145/3797142