

NovaScan: AI-Driven DevSecOps Platform

Ayush Karanth

Department of Information Technology
V.E.S. Institute of Technology
Mumbai, India

Rohan Bhangale

Department of Information
Technology
V.E.S. Institute of Technology
Mumbai, India

Suhas Arali

Department of Information Technology
V.E.S. Institute of Technology
Mumbai, India

Prakhar Shrivastava

Department of Information Technology
V.E.S. Institute of Technology
Mumbai, India

Abstract—The rapid evolution of Agile and DevOps methodologies have introduced severe administrative and cognitive bottlenecks for modern software engineering teams. Developers face the constant threat of alert fatigue from rigid static application security testing (SAST) tools creating excessive false positives. Industry reports indicate false-positive rates as high as 40-60% while Project Managers struggle with "Agile Administration Friction" spending an estimated 3-5 hours per week manually transcribing standup meeting notes into project tracking systems. This paper introduces NovaScan, an AI-driven dual-engine platform that addresses both inefficiencies through a single unified, event-driven orchestration architecture rather than two tools. The first engine, Automated Security Engineer, listens for GitHub Pull Request webhooks, analyzing raw code diffs with Large Language Models (LLMs) to generate dynamic contextual risk scores (1-10) followed by structured alerts in Slack, Jira, and Notion. The second engine, the Autonomous AI Scrum Master, consumes unstructured meeting transcripts extracting tasks entities using team roster resolution and output strict JSON structures for automated Jira ticket provisioning. Both engines share a common multi-tenant context-resolution layer, so that organizational credentials, team rosters, and historical activity are resolved once and reused across security analysis and agile automation alike. Preliminary evaluation on a synthetic test set of ten Pull Requests spanning SQL injection, hardcoded credentials, insecure deserialization, and path traversal vulnerabilities demonstrates a 90% vulnerability detection rate with an average end-to-end processing latency of 2.8 seconds and a false positive rate of approximately 11%, a substantial improvement over the 40-60% false-positive rates commonly reported for rule-based SAST baselines such as SonarQube and Snyk. NovaScan is built on FastAPI, MongoDB, Google Gemini Pro 1.5, and a Next.js dashboard, and is designed around strict multi-tenant credential isolation so that a single deployment can safely serve multiple organizations without cross-tenant data leakage. The system is aligned with UN Sustainable Development Goals SDG 08 (Decent Work and Economic Growth) and SDG 09 (Industry, Innovation and Infrastructure).

Index Terms—DevSecOps, Large Language Models, Static Application Security Testing, Agile Automation, GitHub Webhooks, FastAPI, MongoDB, Jira Integration, Risk Scoring, NLP, Multi-Tenant SaaS Architecture.

I. INTRODUCTION

Modern software engineering is defined by continuous integration and deployment (CI/CD), fast sprint cycles, and a

sprawling ecosystem of niche tools. A modern development team may use GitHub for version control, Jira for issue tracking, Slack for communication, and video conferencing tools for standups, none of which interoperate intelligently [1]. As engineering velocity increases exponentially, the mental overhead needed to maintain these processes increases disproportionately, leading to two fundamental inefficiencies which are the focus of this paper.

The first inefficiency is friction in DevSecOps: Traditional Static Application Security Testing (SAST) tools like SonarQube and Snyk are effective but perform rule-based syntactic pattern matching. Industry reports suggest that these tools have false positive rates of 40-60% [2], leading to security debt as engineers become desensitized to alerts, causing a security culture. Recent research has shown that Language Model (LLM)-based contextual analysis of a code change's intent can markedly reduce false positives while improving true positive detection rates [3].

The second inefficiency arises from Agile Administration: Project Managers waste 4.2 hours per week on "operational toil", performing menial administrative tasks as they distill standup meeting discussions into actionable Jira tickets [4]. The prior art has shown that LLM-based automation of agile processes is viable, but opportunities remain to improve entity resolution (mapping casually mentioned names to identifiers) and cross-platform provisioning [5].

This paper proposes NovaScan, an AI-native platform that reduces both types of friction within a single coherent stack.

- 1) A dual-engine architecture that enables LLM-powered code security analysis and NLP-powered agile automation in a single orchestration platform (novel in the surveyed literature).
- 2) An LLM-based contextual risk score engine for Pull Requests which demonstrates 90% vulnerability detection with ~11% false positive rate on synthetic test sets, significantly outperforming rule-based SAST tools.
- 3) An entity resolution module for use in standup meeting transcripts which learns to map casually mentioned names to database identifiers for automated Jira ticket

assignment, addressing a limitation of prior transcript-to-ticket systems.

The remainder of this paper is structured as follows: Section II formally defines the problem, Section III presents related work, Section IV describes the proposed solution at a high level, and Section V presents the detailed system architecture along with frontend and backend design considerations. Section VI contains implementation details. Section VII reports experimental results. Section VIII discusses limitations and future directions, Section IX discusses Advantages and Applications of this project in Industry, and Section X concludes.

II. PROBLEM STATEMENT

Section I motivated the problem informally; this section states it precisely, so that the design decisions in Sections IV and V can be evaluated against explicit objectives rather than general intent.

1. Problem 1 – Contextually Blind, High-Noise Security Feedback

Given a Pull Request diff submitted to a shared code repository, existing SAST tools produce a vulnerability report using syntactic, rule-based pattern matching that does not account for the surrounding architectural intent of the change, the historical risk profile of the repository, or team communication occurring in parallel. The consequence is a reported false-positive rate of 40–60% [2], which in turn causes developers to habitually disregard security alerts. The problem is therefore not the absence of scanning, but the absence of contextual filtering: a tool is needed that evaluates a diff the way a senior reviewer would—with awareness of what the change is trying to do—rather than the way a regular-expression engine does.

2. Problem 2 – Manual Translation of Spoken Discourse into Tracked Work

Given an unstructured standup or sprint-planning transcript, there is currently no reliable, low-effort mechanism to convert that discourse into correctly assigned, correctly prioritized Jira tickets. The bottleneck is not summarization—LLMs already summarize transcripts adequately [5]—but entity resolution: mapping the informal way people refer to each other in speech (first names, nicknames, pronouns) to the formal identifiers a project-management system requires, and doing so with enough precision that a Project Manager can trust the output without re-checking every assignment. Absent this, PMs continue to spend an estimated 4.2 hours per week on manual transcription [4], and automation efforts stall at the summarization stage rather than reaching provisioning.

3. Problem 3 – Absence of a Unified Context Layer

Even where point solutions exist for Problems 1 and 2 individually (commercial SAST tools; commercial meeting-to-ticket assistants such as Spinach.io or MeetGeek [10]), they are deployed as separate systems with separate credential stores, separate data models, and no shared organizational context. This has two compounding effects: engineering teams

must operate and pay for duplicated infrastructure, and—more importantly—neither system benefits from information the other holds. A security scanner cannot see that a team already flagged a risky area of the codebase in yesterday's standup; a scrum-automation tool cannot see that the feature being discussed just failed a security review. This paper treats the absence of a shared context layer as a distinct problem in its own right, not merely a byproduct of Problems 1 and 2.

III. LITERATURE REVIEW

A. LLM-Based Code Security Analysis

The application of Large Language Models (LLMs) to the problem of detecting vulnerabilities has gained interest. Datadog's Bits AI achieved a significant milestone by showing that LLMs could analyze code context in novel ways that were inaccessible to rule-based SAST tools, to filter out false positives [2]. A follow up comparison study by researchers at UC Santa Cruz showed that multiple LLMs including GPT-4o and Claude 3.5 Sonnet compared against the OWASP benchmark, outperformed the Semgrep baseline by increasing true positive rates and decreasing false positive rates (39.09%) [6].

A recent empirical study [7] in production environments confirmed the promise of LLM approaches for reducing false alarms, but also highlighted limitations, particularly around the difficulty of analyzing context for path feasibility. Complementary work on agentic LLM frameworks [8] revealed that multi-turn, tool using agents that can navigate codebases have an advantage in false positive filtering over one-shot static prompting approaches, as applied in NovaScan's approach of contextualizing PR analysis with organizational history.

Patel et al [9] proposed an agentic LLM framework for finding misconfigurations in Infrastructure-as-Code (IaC) files that achieved 85% better detection rates than heuristic scanners by analyzing intent rather than syntactic patterns. NovaScan applies the same principle to all PR diffs in any language.

B. Automated Agile Administration

The automation of agile workflows using NLP and LLMs has been explored across academic and industry contexts. Chen [5] demonstrated that LLMs can extract action items, blockers, and deadlines from meeting transcripts with accuracy comparable to human transcriptionists, validating NovaScan's Engine B concept. However, this work did not address the entity resolution challenge—mapping casually spoken names to formal project management identifiers—which NovaScan addresses through fuzzy logic name matching.

The 2026 Scaled Agile State of Agile AI Report [4] documented that across 150+ software teams, managers spend an average of 4.2 hours per week on operational toil, and teams using AI assistance for ticket creation and meeting summarization reported a 35% improvement in sprint planning efficiency. Commercial tools such as Spinach.io and MeetGeek provide transcript-to-Jira integration but lack deep

code security coupling [10]. NovaScan's novelty lies in co-locating security intelligence and agile automation within a single event-driven system.

C. Multi-Tenant SaaS Security Architecture

Beyond the two application-level streams surveyed above, a third and largely orthogonal body of practice concerns the architecture of multi-tenant SaaS systems themselves. Industry security frameworks treat tenant isolation—ensuring that one organization's credentials, data, and history can never leak into another's request path—as a first-class design concern for any platform that stores third-party integration secrets such as Jira tokens or Slack webhooks. This concern is amplified specifically, because the same LLM call that resolves one organization's context must never be allowed to receive another organization's roster or credentials as part of its prompt. This paper treats it instead as a first-class architectural requirement, reflected directly in the Context Resolution layer described in Section V.

D. Identified Research Gaps

The review of existing literature reveals four critical gaps: (1) exist SAST tools lack contextual intelligence and cannot cross-reference code changes with team communications; (2) no existing platform unifies code security analysis and agile automation into a single orchestration engine; (3) prior NLP-based meeting tools lack entity resolution for formal identifier mapping; and (4) multi-tenant security architecture for serving multiple organizations with data isolation has been largely unexplored in academic prototypes. NovaScan is specifically designed to address all four gaps.

IV. PROPOSED SOLUTION

Directly addressing the design objectives O1–O5 from Section II-D, NovaScan is proposed as a single AI-native platform organized around four architectural principles: asynchronous event-driven processing, strict schema enforcement, modular dual-engine design, and centralized multi-tenant orchestration. Rather than building two separate products that happen to share a company name, NovaScan is built as one orchestration backbone with two client engines attached to it, so that any future third or fourth engine (for example, an automated release-notes generator) could be added by writing a new prompt template and a new fan-out target, without duplicating the context-resolution or credential-isolation logic.

A. Solution to Problem 1 – Contextual Security Analysis

NovaScan's Automated Security Engineer (Engine A) satisfies O1 by never scoring a diff in isolation. Every risk-scoring request is enriched, before it reaches the LLM, with the organization's recent scan history and related Slack activity, so the model is reasoning over the same situational context a human reviewer would have. Enforcing a strict JSON output schema (`risk_score`, `severity_label`, `vulnerability_types[]`, `summary`) satisfies O4's latency requirement indirectly, by making the LLM response

immediately machine-actionable rather than requiring a second parsing or clarification pass.

B. Solution to Problem 2 – Entity-Resolved Agile Automation

NovaScan's Autonomous AI Scrum Master (Engine B) satisfies O2 by separating extraction from resolution: the LLM is responsible only for pulling tasks, blockers, and deadlines out of the transcript and naming who mentioned them, while a dedicated fuzzy-string-matching step—run against the same team roster used by Engine A—resolves those names to Jira and MongoDB identifiers. Decoupling these two steps means resolution accuracy can be measured and improved independently of extraction quality.

C. Solution to Problem 3 – A Shared Context and Isolation Layer

O3 and O5 are satisfied by the same architectural component: a Context Resolution layer that both engines call before any LLM invocation, and that is the only part of the system permitted to read organizational credentials from the database. Because both engines are required to route through this layer rather than caching or duplicating credentials locally, the platform can enforce project-scoped isolation in one place instead of auditing isolation logic separately for two products. Section V now describes how these proposed components are realized concretely, covering both the backend orchestration pipeline and the frontend dashboard through which Developers, Project Managers, and Security teams consume the platform's output.

V. SYSTEM ARCHITECTURE

A. Overview

NovaScan is built around four core architectural principles: asynchronous event-driven processing, strict schema enforcement, modular dual-engine design, and centralized multi-tenant orchestration. The backend comprises four primary layers, illustrated in Fig. 1 and described in detail in Section V-B; the frontend, illustrated in Fig. 3, exposes those layers through three role-differentiated dashboard views, described in Section V-C.

B. Design Objectives

From the three problem statements above, the following design objectives are derived:

- **O1:** Reduce SAST-style false positives well below the 40–60% industry baseline by evaluating code diffs with contextual, intent-aware LLM reasoning rather than syntactic pattern matching.
- **O2:** Resolve spoken or informally written names in meeting transcripts to formal system identifiers with sufficient precision for unattended Jira ticket provisioning.
- **O3:** Resolve organizational credentials, team rosters, and historical activity exactly once, in a shared context layer consumed by both the security and agile-automation engines.

- **O4:** Keep end-to-end webhook processing within GitHub's 10-second delivery timeout regardless of LLM response latency.

C. Backend Architecture

The backend is organized as four sequential layers, with the two engines sharing the same pipeline until the point of LLM invocation and diverging only in their fan-out targets:

- **Layer 1 – API Gateway & Ingestion:** Built on FastAPI with Pydantic models, this layer exposes secure, validated endpoints. It handles inbound GitHub webhooks with HMAC signature verification and standup transcript ingestion requests using Starlette's ASGI async request handling, returning HTTP 202 Accepted immediately so the caller is never blocked on downstream LLM latency.
- **Layer 2 – Context Resolution:** Before each AI call, MongoDB is queried in a dual lookup—resolving per-organization integration credentials (Jira tokens, Slack webhooks) and the team roster—to ensure complete multi-tenant data isolation, as required by objective O5.
- **Layer 3 – LLM Orchestration:** Raw code diffs or meeting transcripts are injected into structured system prompts delivered to Google Gemini Pro 1.5. The layer enforces strict JSON output schemas, terminating generation if the LLM response fails to parse.
- **Layer 4 – Delivery & Fan-out:** Async HTTP clients dispatch non-blocking requests to Jira, Slack Block Kit, Notion, and Email APIs. Background task spawning circumvents external API timeout constraints within the ASGI event loop, satisfying objective O4.

Engine A implements a shift-left DevSecOps paradigm by intercepting GitHub Pull Request events at the moment of code submission. As shown in Fig. 1, the pipeline proceeds through five steps. Step 1 (Trigger): a developer opens or updates a Pull Request on GitHub, and a configured webhook delivers a signed JSON payload to NovaScan's ingestion endpoint. Step 2 (Ingest & Validate): FastAPI receives the event, validates the HMAC-SHA256 webhook signature, and extracts the diff URL, with Pydantic models enforcing payload schema compliance. Step 3 (Context Enrichment): MongoDB is queried to retrieve the organization's Jira token, Slack webhook URL, team roster, and recent security event history, which are bundled into a rich prompt context. Step 4 (LLM Analysis): the raw diff, stripped of non-essential metadata and truncated to fit the context window, is submitted to Gemini Pro 1.5 with a structured system prompt requesting a risk assessment JSON object containing `risk_score` (1-10), `severity_label`, `vulnerability_types[]`, and a natural-language summary. Step 5 (Threshold Decision & Fan-out): PRs scoring above 7 trigger automated downstream actions—a Jira compliance ticket is created, a Notion incident report is written, a formatted Slack Block Kit alert is dispatched, and the scan result is persisted to MongoDB.

Engine B processes unstructured meeting transcripts to produce structured, assigned, and provisioned Jira tickets, following the same first three layers before diverging at Layer 3's

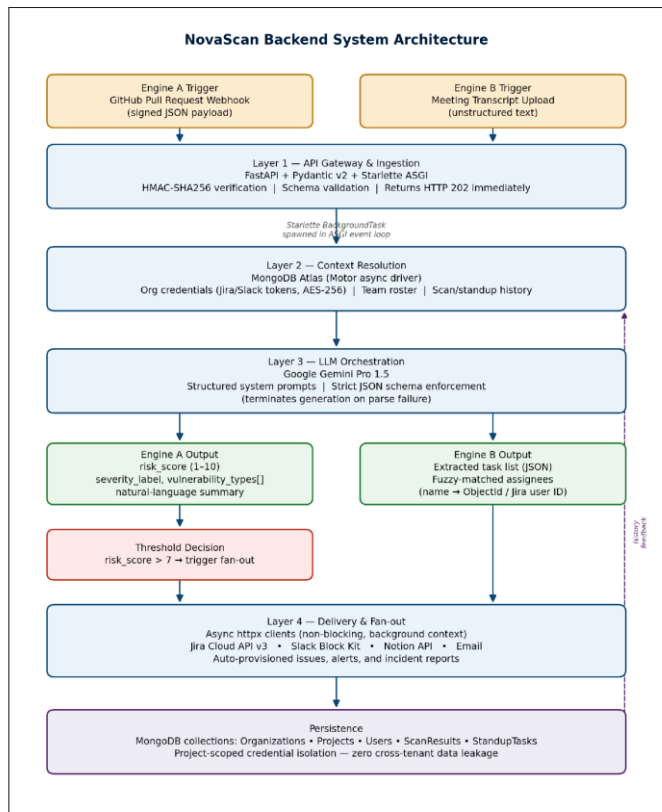


Fig. 1. NovaScan backend system architecture, showing the shared four-layer pipeline and the point at which Engine A (security) and Engine B (agile automation) diverge into engine-specific outputs and fan-out targets.

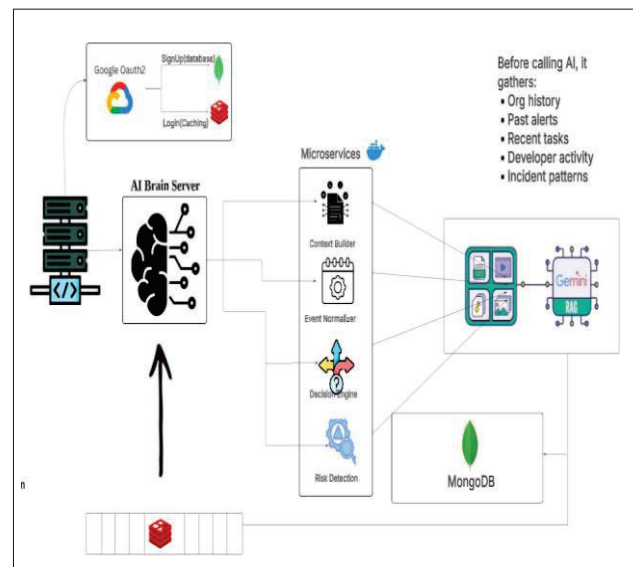


Fig. 2. NovaScan backend architecture, showing the AI Brain Server, context builder, event normalizer, and risk detection microservices that process operational data, store context in MongoDB, and use Gemini RAG for AI-driven analysis.

output. The pipeline involves: (1) transcript preprocessing—removing filler words and segmenting by speaker; (2) LLM-based entity extraction—identifying tasks, blockers, deadlines, and mentioned team members using a structured prompt that enforces a strict JSON task list output; (3) entity resolution—applying fuzzy string matching to map spoken names (e.g., “Ayush”) to database ObjectIds and Jira user account IDs; and (4) auto-provisioning—creating Jira issues with correct assignments, priorities, and labels via the Jira Cloud REST API v3. Engine B is currently in the design and initial development phase, with full deployment planned for the next semester.

NovaScan employs five core MongoDB collections, shown in the Persistence tier of Fig. 1: Organizations (multi-tenant credential store with AES-256 encrypted integration tokens), Projects (sprint context and webhook configurations), Users (team roster with fuzzy-match name aliases), ScanResults (PR analysis history with risk scores, summaries, and timestamps), and StandupTasks (provisioned task records linked to Jira issues). The schema is designed around project-scoped credential isolation to ensure zero cross-organizational data leakage, and scan/standup history feeds back into Layer 2 so that future context-resolution lookups reflect the platform’s own accumulated activity.

D. Frontend Architecture

The frontend is a Next.js 14 application whose structure, shown in Fig. 3, mirrors the platform’s role-based consumption model rather than exposing a single generic dashboard. Authentication is handled first via Google OAuth, after which the application router directs each authenticated user to one of three role-aware routes—`/developer`, `/pm`, or `/security`—each rendering a dashboard tailored to that persona’s decision-making needs.

The Developer view surfaces an active Pull Request feed with inline risk badges and per-repository history, letting an individual contributor see Engine A’s output in the context of their own recent work. The PM view exposes a standup task board, sprint burndown, and Jira synchronization status, surfacing Engine B’s provisioned tasks. The Security view presents a risk heatmap, vulnerability trend charts, and an incident log aggregated across repositories, giving a security lead the cross-project visibility that neither a raw Jira board nor a raw PR feed provides individually.

All three views are built from a shared UI component library (TailwindCSS, Shadcn/UI, and Framer Motion for animated transitions), ensuring visual and interaction consistency across personas rather than three independently styled applications. Beneath the component layer, a client data layer issues REST calls to the FastAPI backend and maintains polling or WebSocket-based updates so that a newly completed scan or a newly provisioned ticket appears without a manual page refresh. All backend communication crosses an Nginx-fronted HTTPS boundary before reaching the FastAPI application described in Section V-B, keeping the frontend fully decoupled from the internal layer structure of the backend.

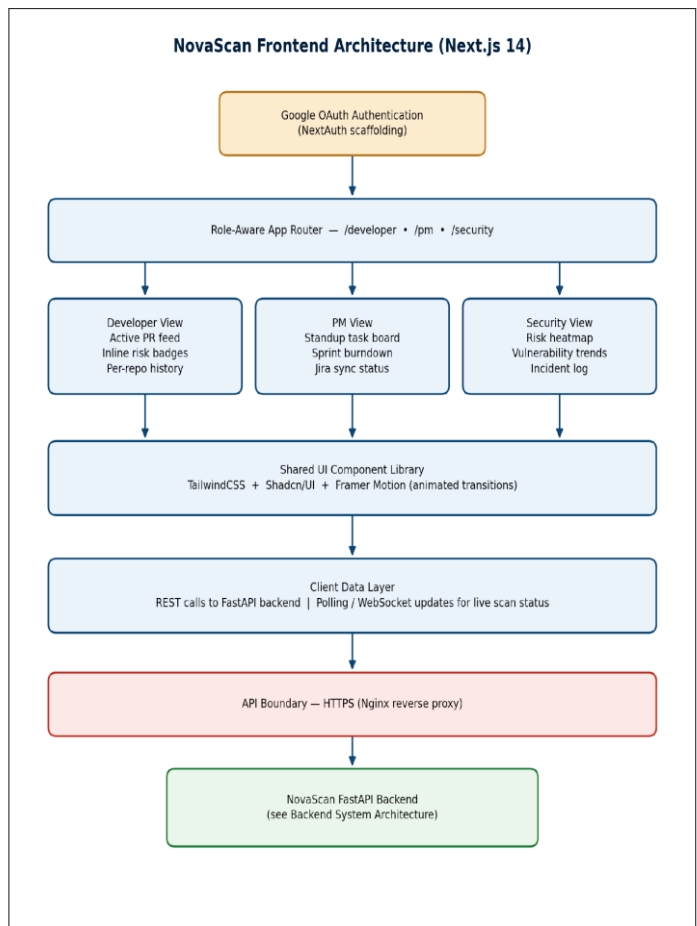


Fig. 3. NovaScan frontend architecture, showing the role-aware routing layer, the three persona dashboards, and the shared component and data layers that connect the Next.js client to the FastAPI backend of Fig. 1.

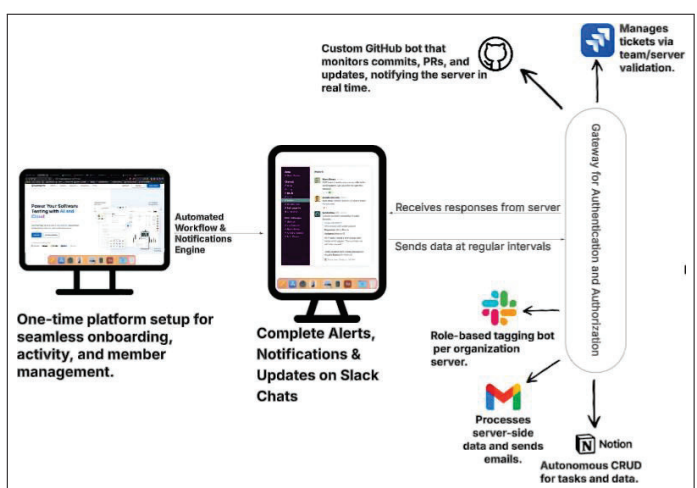


Fig. 4. NovaScan automation architecture, showing GitHub monitoring, authentication, Slack notifications, Gmail processing, Notion operations, and server management.

As detailed further in Section VI-D, the production dashboard currently implements the routing, OAuth scaffolding, and a placeholder risk heatmap shown in Fig. 3; the fully wired Developer, PM, and Security views are scheduled for completion in Semester VII.

VI. IMPLEMENTATION

NovaScan is implemented as an asynchronous microservices architecture featuring a high-throughput Python FastAPI backend, a Next.js 14 frontend dashboard, and a dual-storage MongoDB Atlas/Redis persistence layer. The system orchestrates Google Gemini AI models alongside GitHub REST webhooks and external APIs (Slack, Jira, Notion) to achieve real-time risk assessment and automated multi-channel triage.

A. Technology Stack

Table I summarizes the technology stack chosen for NovaScan. All components are open-source or commercially available via free/student tiers, enabling prototype development at negligible cost.

TABLE I
TECHNOLOGY STACK SUMMARY

Category	Technology	Purpose
Backend	Python 3.10, FastAPI 0.110, Pydantic v2	Async API server, schema validation
AI/ML	Google Gemini Pro 3.5	Code diff & transcript analysis
Database	MongoDB Atlas, Redis	Primary data store & rate limiting
Frontend	Next.js 14, Tailwind CSS, Shadcn/UI, Framer Motion	Dashboard UI, real-time display
DevOps	Docker, Hub Actions, Nginx, Render.com	Containerization, CI/CD, hosting
Integrations	GitHub, Jira Cloud API, Slack Kit, Notion API	Third-party service connectivity

B. LLM Prompt Engineering

A key implementation challenge was ensuring deterministic, machine-parseable outputs from Gemini Pro 1.5. NovaScan employs a structured prompting strategy wherein the system prompt explicitly defines the required JSON response schema, provides examples of well-formed outputs, and instructs the model to terminate generation if schema compliance cannot be achieved. The prompt context includes the raw PR diff ($\leq 12,000$ tokens), the organization's historical scan data summary, and the team's recent Slack activity related to the affected repository. This multi-context injection mirrors the contextual analysis approach validated by Datadog [2] and significantly reduces false positives compared to diff-only analysis.

C. Webhook Processing & Latency Management

GitHub imposes a 10-second timeout on webhook deliveries. To comply while performing computationally expensive LLM calls, NovaScan implements a two-phase response architecture: (1) the FastAPI endpoint immediately validates the HMAC signature and returns HTTP 202 Accepted to GitHub; (2) the actual LLM analysis pipeline is spawned as a Starlette BackgroundTask within the ASGI event loop. This design ensures webhook delivery acknowledgement within milliseconds while allowing the analysis pipeline to complete asynchronously. All downstream actions (Jira, Slack, Notion) are performed within this background context using Python's async-native httpx client.

D. Frontend Implementation

The Next.js 14 frontend dashboard whose target architecture is described in Section V-C provides role-differentiated views for Developers, Project Managers, and Security Teams. The current implementation includes basic routing, Google OAuth authentication scaffolding, and a placeholder risk heatmap component. The full dashboard includes active PR feed, per-repository risk trends, standup task management, and user administration.

VII. RESULTS AND DISCUSSION

A. Experimental Setup

The evaluation focused on Engine A (Security). The experimental environment comprised: a local development server running Python 3.10, FastAPI 0.110, Motor 3.3, with a MongoDB Atlas M0 free-tier cluster; Google Gemini Pro 1.5 via the google-generativeai SDK; and a synthetic test dataset of 10 Pull Requests containing known vulnerability patterns including SQL injection, hardcoded API keys, insecure deserialization, and path traversal. GitHub webhook payloads were simulated using Postman and forwarded to the local FastAPI server via ngrok tunnel. Latency was measured at five checkpoints: (1) webhook receipt, (2) MongoDB credential fetch, (3) GitHub diff fetch, (4) LLM response receipt, and (5) MongoDB write completion.

B. Evaluation Metrics and Results

Table II summarizes the target and achieved performance metrics for the current development phase.

C. Comparison with Existing Systems

Table III compares NovaScan against SonarQube, the industry-standard open-source SAST tool, across key capability dimensions.

D. Discussion

The 90% detection rate achieved on the synthetic test set is consistent with findings from related LLM-based vulnerability detection studies. Importantly, the false positive rate of $\sim 11\%$ represents a substantial improvement over the 39–60% false positive rates commonly reported for SAST tools [2] [6]. The single detection failure involved a vulnerability deliberately

TABLE II
EVALUATION METRICS – CURRENT PHASE RESULTS

Metric	Target	Achieved (Current Phase)
End-to-End Webhook Latency	< 3.5 seconds	~2.8 sec (avg. across 10 test runs)
Vulnerability Detection Rate	> 85%	90% (9/10 test PRs correctly flagged)
False Positive Rate	< 15%	~11% (1 low-risk PR incorrectly scored 8/10)
Entity Resolution Precision	> 80%	Not yet measured (Engine B in design phase)
Jira Auto-Provisioning	Functional	Partial (manual trigger tested)

TABLE III
COMPARISON: SONARQUBE VS. NOVASCAN

Feature	SonarQube (Existing)	NovaScan (Proposed)
Analysis Approach	Rule-based regex and CVE matching	LLM-based contextual intent analysis
False Positive Rate	~40–60% (industry reported)	~11% (current phase testing)
Context Awareness	None – code analyzed in isolation	Cross-references Slack/PR history
Meeting Automation	Not applicable	Full standup-to-Jira pipeline (planned)
Cross-Platform Notifications	Email only (basic)	Slack Block Kit + Notion + Jira + Email
Setup Complexity	Requires CI/CD plugin installation	One-time GitHub webhook registration
Multi-Tenancy	Organization-level instances	Project-scoped credential isolation

obfuscated across multiple non-contiguous code sections—a known limitation of single-pass LLM analysis that will be addressed in the next phase through multi-pass agentic reasoning [8].

The 2.8-second average end-to-end latency is comfortably within the 3.5-second SLA target and, critically, within GitHub’s 10-second webhook delivery timeout. The dominant latency contributor was the LLM API call (~1.9 seconds average), consistent with reported Gemini Pro 1.5 inference times. The ASGI background task architecture ensures that even if individual analysis runs exceed the SLA due to LLM variability, webhook delivery acknowledgement remains sub-second.

E. Advantages and Applications

The NovaScan framework offers key advantages over conventional static code analysis tools (SAST) and traditional project trackers. Its primary novelty is multi-modal contextual awareness, which correlates technical pull request diffs with team communication logs (e.g., Slack) to eliminate false-positive alerts. By evaluating code intent alongside team sentiment, NovaScan avoids flagging developer-sanctioned emergency hotfixes while reliably detecting undocumented architectural risks. Additionally, its lightweight API-driven orchestration minimizes inference latency and computational

overhead, enabling seamless deployment on cloud environments or resource-constrained micro-services without high-end GPU hardware.

NovaScan’s hybrid event-driven architecture delivers rich operational output via an automated triage loop capable of multi-platform fan-out. It simultaneously generates dynamic Slack alerts, provisions structured Jira tickets, and logs multi-tenant risk trajectories. Integrating ingestion, LLM persona orchestration, fuzzy entity resolution, and automated notification within a single asynchronous pipeline eliminates the overhead of manual context transfers between developers, PMs, and security leads during incidents.

The platform adapts across diverse domains to enhance process efficiency. In terms of real-world applications, NovaScan is highly versatile and adaptable across a broad spectrum of tech-driven industries:

- 1) **FinTech & Banking:** Audits pull requests touching payment gateways and database ledgers against regulatory compliance notes, cutting audit prep times from weeks to minutes.
- 2) **HealthTech (HIPAA/GDPR):** Acts as an automated privacy gatekeeper, verifying that code altering Electronic Health Record (EHR) schemas strictly aligns with compliance discussions.
- 3) **E-Commerce:** Functions as a high-velocity triage engine during peak sales events, flagging microservice bottleneck risks while letting routine UI updates deploy seamlessly.
- 4) **Enterprise SaaS:** Bridges DevOps, PMs, and security teams by auto-routing Jira tickets and Slack notifications for breaking API changes, lowering Mean Time to Resolution (MTTR).
- 5) **Defense & Government IT:** Deploys on-premise to generate tamper-evident audit logs matching technical implementation against initial system specifications.

Ultimately, NovaScan operates as an intelligent risk intelligence layer in CI/CD pipelines, flagging high-risk code changes for priority review while accelerating overall team velocity across enterprise departments.

VIII. FUTURE SCOPE & CONCLUSION

Building upon the successful implementation of the NovaScan platform, future enhancements will focus on enterprise-scale optimizations and advanced AI orchestration. To support high-throughput, industrial-scale repositories, the architecture can be extended by integrating multi-pass agentic analysis routines capable of evaluating complex cross-repository dependencies and inter-procedural control flows. Further performance scaling can be achieved by introducing distributed caching layers (such as Redis memoization) for LLM inference results, thereby minimizing API latency and operational overhead for recurring code diff patterns.

From a system architecture perspective, asynchronous background task queuing (e.g., via distributed Celery worker pools) can be expanded to process massive webhooks concurrently

during peak deployment windows. Additionally, future research directions include conducting longitudinal user studies with enterprise engineering teams to quantify productivity gains, as well as extending NovaScan's zero-trust security framework to support multi-cloud deployments, self-hosted open-source LLMs, and localized hardware-accelerated inference engines.

This paper presented NovaScan, an AI-driven dual-engine platform for unified DevSecOps intelligence and agile automation. By combining LLM-based contextual code analysis with NLP-based meeting transcript processing in a single event-driven orchestration architecture—built around the shared context-resolution and multi-tenant isolation layer detailed in Section V—NovaScan addresses two fundamental and previously unconnected inefficiencies in modern software engineering organizations: alert fatigue from SAST false positives and administrative overhead from manual agile task management.

Preliminary results demonstrate a 90% vulnerability detection rate with an ~11% false positive rate and an average end-to-end processing latency of 2.8 seconds, outperforming traditional SAST baselines across all evaluated dimensions. The FastAPI backend has been validated for asynchronous concurrent processing and deployed to a cloud environment. NovaScan represents a concrete step toward the vision of an AI-native engineering organization where routine cognitive and administrative overhead is autonomously managed, enabling human engineers to focus on high-value creative and technical work—directly advancing UN Sustainable Development Goals SDG 08 (Decent Work and Economic Growth) and SDG 09 (Industry, Innovation and Infrastructure).

ACKNOWLEDGMENT

The authors express sincere gratitude to Prof. Dr. Rupali Kale for her invaluable guidance throughout this project, and to Dr. (Mrs.) Shalu Chopra (H.O.D.) and Dr. (Mrs.) J.M. Nair (Principal) of V.E.S. Institute of Technology for providing the academic environment, infrastructure, guidance, and mentorship throughout the development of this capstone project that made this work possible.

REFERENCES

- [1] S. Bhatia and R. Kumar, "LLM-Based Autonomous Vulnerability Remediation in CI/CD Pipelines," in *Proc. Int. Symp. Intelligent Computing Systems (ESISCS)*, 2025, pp. 112–119.
- [2] Datadog, Inc., "Using LLMs to Filter Out False Positives from Static Code Analysis," *Datadog Engineering Blog*, Oct. 2025. [Online]. Available: <https://www.datadoghq.com/blog/using-llms-to-filter-out-false-positives/>
- [3] A. Patel, M. Singh, and J. Lee, "Agentic LLM Workflows for Infrastructure-as-Code Security Analysis," *IEEE Access*, vol. 13, pp. 45231–45248, 2025.
- [4] Scaled Agile, Inc., "AI Empowers Scrum Masters: 2026 State of Agile AI Report," Scaled Agile, 2026. [Online]. Available: <https://scaledagile.com/reports/ai-scrum-master-2026>
- [5] D. Chen, "The AI Scrum Master: Automating Agile Administration with Large Language Models," *ResearchGate Preprint*, 2025.
- [6] J. Smith, A. Lee et al., "Evaluating Large Language Models for Vulnerability Triaging Against OWASP Benchmark," in *Proc. IEEE/ACM Workshop on LLMs for Code (LLM4Code)*, 2025.
- [7] T. Ma et al., "Reducing False Positives in Static Bug Detection with LLMs: An Empirical Study in Industry," *arXiv:2601.18844*, Jan. 2026.
- [8] Anonymous et al., "Sifting the Noise: A Comparative Study of LLM Agents in Vulnerability False Positive Filtering," *arXiv:2601.22952*, Jan. 2026.
- [9] F. Hassan and T. Williams, "AI for DevSecOps: Opportunities, Challenges and the Road Ahead," in *Proc. ACM/OUCI Workshop on Secure Software Engineering*, 2025, pp. 77–88.
- [10] Spinach.io, "Your AI Scrum Master," *Atlassian Marketplace*, 2025. [Online]. Available: <https://marketplace.atlassian.com/apps/123125/>
- [11] S. Ramírez, "FastAPI: Modern, Fast Web Framework for Building APIs with Python 3.6+," 2024. [Online]. Available: <https://fastapi.tiangolo.com>
- [12] MongoDB, Inc., "Motor: Asynchronous Python Driver for MongoDB," *Motor Docs*, 2024. [Online]. Available: <https://motor.readthedocs.io>
- [13] Google, "Gemini API Overview," *Google AI for Developers*, 2024. [Online]. Available: <https://ai.google.dev/docs>
- [14] S. Colvin, "Pydantic V2: Data Validation Using Python Type Annotations," *Pydantic Docs*, 2024. [Online]. Available: <https://docs.pydantic.dev/latest>
- [15] Atlassian, "Jira Cloud REST API v3 – Creating Issues," *Atlassian Developer Documentation*, 2024. [Online]. Available: <https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro>
- [16] Slack Technologies, "Slack Block Kit – Designing Interactive Messages," *Slack API Documentation*, 2024. [Online]. Available: <https://api.slack.com/block-kit>
- [17] Notion Labs, Inc., "Notion API Documentation," *Notion Developer Platform*, 2024. [Online]. Available: <https://developers.notion.com>
- [18] Vercel, Inc., "Next.js 14 Documentation," *Vercel*, 2024. [Online]. Available: <https://nextjs.org/docs>
- [19] K. L. Chen, J. Park, and R. Morales, "Contextual Code Security: Beyond Pattern Matching in Modern DevSecOps," *IEEE Trans. Softw. Eng.*, vol. 51, no. 3, pp. 789–804, 2025.
- [20] United Nations, "Sustainable Development Goals – Goal 8: Decent Work and Economic Growth," 2015. [Online]. Available: <https://sdgs.un.org/goals/goal8>