SentinelX AURAPRIME: An AIDriven DevSecOps Intelligence Platform for Software Supply Chain Risk Management Through Developer Digital Twin and GTI++ Unified Trust Scoring

Mohamed Ashish Roshan H[1], Mohamed Aasif S[2], Mohamed Arish S[3], Manoj Kumar M[4]
Ms. S. Selvashanthi[5]
[1,2,3,4]*Department of Computer Science and Engineering, M.I.E.T. Engineering College, Tiruchirappalli, India*
[5]*Under the guidance, AP/CSE M.I.E.T. Engineering College, Tiruchirappalli, India*

***Abstract*—Modern software organizations face critical challenges from security tool fragmentation, with enterprises deploying 10–15 disconnected tools across the Software Development Life Cycle (SDLC), contributing to an average breach detection time of 280 days and costs exceeding $4.45 million per incident. Existing solutions analyze only code artifacts while ignoring developer behavior the human factor behind 85% of security breaches. This paper presents Sentine lX AURAPRIME, an AI driven DevSecOps intelligence platform unifying 26 security modules across three architectural tiers Core Security, Intelligence, and Frontier into a cohesive, production grade system with 55 REST API endpoints. Two novel research contributions are introduced: (1) the Developer Digital Twin (DDT), an ML powered behavioral analysis framework that constructs security risk profiles from developer coding patterns through Abstract Syntax Tree (AST) parsing across six behavioral dimensions, achieving 76.4% risk prediction accuracy; and (2) the GTI++ (Global Trust Index Plus Plus) algorithm, an 8dimensional weighted risk scoring model that synthesizes code quality, developer behavior, pipeline security, container integrity, runtime posture, network exposure, cloud identity, and SOC incident metrics into a single explainable trust score with sub300ms computation. Experimental evaluation on 150 opensource repositories demonstrates 94.2% vulnerability detection with 8.7% false positives, SOAR based automated remediation achieving 1800× speedup, Neural WAF showing 22.8% improvement over regex-based detection, and zero click auto healing in ∼15 seconds. The platform integrates Claude 3.5 Sonnet with a three tier LLM fallback ensuring 100% availability. Comparative analysis against Snyk, SonarQube, and GitLab SAST demonstrates 70% cost reduction with seven unique capabilities absent from all competing platforms.**

***Index Terms*—DevSecOps, Developer Digital Twin, GTI++ Risk Scoring, AI Powered Security, SOAR Automation, Behavioral Fingerprinting, Shift Left Security, Software Supply Chain, Large Language Models, Auto Healing**

INTRODUCTION

The rapid adoption of DevOps methodologies has fundamentally transformed software delivery, enabling organizations to achieve continuous integration and deployment (CI/CD) at unprecedented velocity. Modern engineering teams at organizations such as Netflix, Amazon, and Google deploy code changes thousands of times per day, compressing release cycles from quarterly milestones to sub hourly deployments. However, this acceleration has introduced systemic security challenges that traditional waterfall era security practices designed for quarterly release cadences with dedicated security review phases are structurally incapable of addressing [1]. The DevSecOps paradigm emerged in response, seeking to embed security as an integral concern throughout every phase of the SDLC rather than treating it as a post development gate. Yet practical DevSecOps implementation remains fragmented across disconnected point solution tools, each addressing a narrow slice of the overall attack surface

[5]. The scale and severity of the problem are acute and well documented. According to the IBM 2024 Cost of a Data Breach Report [7], organizations take an average of 280 days to identify and contain a security breach, with mean costs reaching $4.45 million per incident. Furthermore, the report reveals that breaches identified after 200 days cost 35% more than those caught within the first 100 days, with costs scaling to $4.9M for late detected incidents versus $3.1M for early detected ones. This creates a powerful economic argument for shift left security practices that detect vulnerabilities as close to the point of introduction as possible. The 2024 Verizon Data Breach Investigations Report (DBIR) [8] further reveals that 85% of all security breaches involve a human element encompassing social engineering, credential misuse, and human error in code and configuration. Despite this finding, no existing commercial or academic security tool models developer behavior as a first-class security variable. All current tools analyze downstream artifacts (source code, container images, network configurations) but ignore the upstream human agent responsible for creating those artifacts. This fundamental blind spot constitutes the primary motivation for the research presented in this paper.

The contemporary security tooling landscape compounds these challenges through fragmentation and cognitive overload. Enterprise security teams routinely operate 10–15 separate tools: Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Software Composition Analysis (SCA), container vulnerability scanners, secrets detection engines, Infrastructure as Code (IaC) analyzers, Identity and Access Management (IAM) auditors, compliance checkers, Web Application Firewalls (WAFs), and Security Information and Event Management (SIEM) platforms. Each tool generates its own alerts, maintains its own dashboard, and reports findings in incompatible formats. The result is a fragmented operational picture that prevents holistic risk assessment and delays executive decision-making.

A. Problem Statement

We identify six critical, interconnected challenges in the current DevSecOps landscape that collectively undermine software security:

1. Tool Fragmentation: Enterprises deploy 10–15 disconnected security tools, incurring $250K–$500K annual licensing and infrastructure costs. Integration overhead consumes 20–30% of security engineering bandwidth, and teams must context switch between 5–8 separate consoles to form a holistic risk view. This fragmentation creates gaps where vulnerabilities fall between tool boundaries and are never detected.
2. Delayed Detection: The 280day average breach identification window [7] renders purely reactive security approaches catastrophically expensive. Attackers leverage this detection gap to establish persistence, move laterally across systems, exfiltrate data, and compromise additional infrastructure, with costs scaling super linearly with detection delay.
3. Alert Fatigue and False Positives: Commercial SAST and DAST tools exhibit false positive rates of 40–60%, generating thousands of low fidelity alerts per scan cycle. Research shows that this volume overwhelms security teams, causing 27% of genuinely critical alerts to be dismissed or deprioritized. Attackers can exploit this pattern by embedding real vulnerabilities among common false positive signatures.
4. No Unified Risk View: No existing platform synthesizes cross dimensional security risk into a single, actionable, and explainable metric. A container vulnerability, a leaked credential, and an over permissioned IAM role may each appear low risk individually, but together create a critical attack chain that siloed tools cannot detect.
5. Manual Incident Response: Average response times of 15 minutes for critical vulnerability blocking and 4 hours for patch deployment create exploitation windows that automated adversaries can leverage in seconds. Manual processes requiring human triage, analysis, patch development, testing, and deployment are fundamentally incompatible with the velocity of modern automated attacks.
6. Developer Blind Spots: All existing tools analyze code artifacts but fail to model the *developer* the human agent behind 85% of security incidents [8]. A developer who consistently writes insecure code due to poor security practices represents a systematic, recurring risk that static code analysis alone cannot address.

B. Proposed Solution and Contributions

We present SentinelX AURAPRIME, a unified AI driven DevSecOps intelligence platform that addresses all six challenges through four key research contributions:

1. Developer Digital Twin (DDT): A novel ML based behavioral fingerprinting system that constructs security risk profiles from developer coding patterns across six dimensions using Abstract Syntax Tree (AST) analysis, achieving 76.4% prediction accuracy a 46.6% relative improvement over random baseline using a Random Forest classifier trained on features from 7,500 developer repository pairs (Section V).
2. GTI++ Algorithm: An 8dimensional weighted scoring model producing a single explainable trust score with five deterministic decision thresholds (allow/warn/review/block/escalate) and sub300ms computation, with built-in explainability through dimensional gap analysis addressing the "black box" barrier identified by Brown et al. [3] (Section VI).
3. Unified 26Module Platform: Three tier architecture (Core Security, Intelligence, Frontier) with 55 REST end points, SOAR based automation achieving 1800× incident response speedup, zero click auto healing in ∼15 seconds, and Neural WAF with +22.8% detection improvement through dual engine (regex + AI) architecture.
4. Three Tier AI Fallback: Claude 3.5 Sonnet [17] (cloud, 200K context) → Ollama (local, air gapped) → Deterministic Simulation (hash based, offline), ensuring 100% operational availability across cloud, air gapped, and demonstration environments.

C. Paper Organization

Section II reviews six base papers with gap analysis. Section III presents the system architecture. Section IV details all 26 modules. Sections V and VI formalize the novel algorithms. Section VII presents experimental evaluation. Section VIII discusses findings and limitations. Sections IX and X outline future work and conclusions.

II. RELATED WORK

A. AI augmented DevSecOps Pipelines

Kumar et al. [1] proposed AI augmented DevSecOps pipelines for cloud native service-oriented architectures, demonstrating that embedding ML classifiers at build and deploy stages within CI/CD workflows reduces false positive rates by 18% compared to rule based approaches, achieving 82% overall detection accuracy. Their architecture introduces the concept of "security gates" automated checkpoints where ML models evaluate deployment readiness based on scan results. This represents a significant advance over purely rule based gating. However, their pipeline operates as a single dimensional binary gate (pass/fail) without dimensional risk decomposition, providing no granularity between "safe" and "unsafe." The framework lacks developer behavioral modeling entirely, treats all code identically regardless of author, and provides no automated remediation capability when a gate blocks deployment, manual intervention is required. SentinelX extends this work by replacing binary gates with multidimensional GTI++ scoring (5 graduated decisions), introducing DDT for proactive behavioral prediction, and adding SOAR automation for sub second response.

B. AI Code Generators for Secure Code Synthesis

Zhang et al. [2] conducted a systematic evaluation of Large Language Models for generating security hardened code across five programming languages and eight vulnerability categories. Their findings demonstrate that security constrained prompting injecting OWASP rules, authentication patterns, and encryption requirements into the LLM context window reduces vulnerability rates in generated code from 34% to 11%. However, they identify a critical residual 23% vulnerability rate even with optimized prompt strategies, concluding that "no prompt engineering strategy alone can guarantee vulnerability free code generation." This fundamental limitation motivates a key architectural decision in SentinelX: the AI Code Forge module (M1) chains LLM code generation with automated post generation SAST scanning via Sem grep [15], applying 47 OWASP Top 10 [11] security constraints to the generation context while independently validating all outputs through static analysis effectively closing the

residual vulnerability gap through integrated verification rather than relying solely on prompt engineering.

C. LLMs in Cybersecurity: Dual Use Risks and Explainability

Brown et al. [3] surveyed dual use risks of LLMs in cybersecurity, conducting interviews with 47 security practitioners across 23 organizations. Their central finding is that the "black box" problem represents the primary barrier to enterprise adoption of AI powered security tools: 73% of AI generated security recommendations are rejected by security teams when the underlying reasoning is opaque, rendering the AI investment counterproductive. They propose that explainable AI (XAI) is not merely desirable but a prerequisite for trustworthy security automation in enterprise contexts. This finding directly shapes the explainability architecture of GTI++. Unlike commercial scoring systems (e.g., Snyk Priority Score, SonarQube Quality Gate) that output opaque numeric values without dimensional decomposition, GTI++ provides transparent gap analysis: for each computed trust score, the system ranks all eight contributing dimensions by their gap contribution ($w_j \cdot (100 - S_j)$) and generates specific, actionable remediation guidance for the top5 risk factors enabling security teams to understand *why* a decision was made and *what* to fix first.

D. Tamper Evident Log Intelligence

Smith et al. [4] proposed DevSecLogs, an AI powered tamper evident log intelligence framework implementing event correlation and anomaly detection for CI/CD pipeline logs. Their temporal correlation approach using 10minute sliding windows achieves 88.3% anomaly detection accuracy with a 4.2% false positive rate. The framework introduces cryptographic hash chaining for tamper detection, ensuring that any modification to historical log entries is immediately detectable. However, DevSecLogs focuses exclusively on detection without providing any automated response capability when anomalies are detected, the system generates alerts that require manual investigation and remediation. SentinelX's SOCLite module (M11) adapts and extends this approach: it reduces the sliding window to 5 minutes for faster temporal correlation, implements five stage incident lifecycle management (new → investigating → contained → resolved → closed), and critically adds automated response via the SOAR engine (M12) closing the detectto respond loop that DevSecLogs leaves open.

E. Threat Modeling for DevSecOps Environments

Johnson et al. [5] presented a comprehensive threat taxonomy for DevSecOps environments, systematically mapping 24 distinct attack surfaces to the STRIDE classification (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) across four SDLC phases. Their work provides the most thorough threat enumeration for DevSecOps to date, identifying specific attack vectors at each pipeline stage. SentinelX implements dedicated modules addressing each STRIDE category: Spoofing (M8: IAM Security), Tampering (M3: CI/CD Shield with integrity checks), Repudiation (M11: SOCLite audit trails), Information Disclosure (M7: Secrets Management), Denial of Service (M23: Neural WAF), and Elevation of Privilege (M8: IAM + M22: Blast Radius). This constitutes the first platform to achieve complete STRIDE coverage through a unified architecture.

F. Zero Trust with MITRE ATT&CK Integration

Lee et al. [6] investigated integrating Zero Trust architecture [9] with the MITRE ATT&CK Matrix [16] for cyber resilience, proposing continuous trust scoring based on real-time behavioral signals with automated policy enforcement when trust scores fall below configurable thresholds. Their framework achieves 91% accuracy in trust/deny classification but operates as a binary system without risk graduation every evaluation result in either full trust or complete denial, with no intermediate states. SentinelX's GTI++ extends this with five tier graduated decisions (allow / warn / review / block / escalate), implementing the Zero Trust principle of "never trust, always verify" with 8dimensional continuous verification and proportionate response escalation.

III. SYSTEM ARCHITECTURE

SentinelX employs a four-layer modular monolith architecture (Fig. 1) designed to satisfy three competing requirements: security depth

(comprehensive coverage across all SDLC phases), operational simplicity (single deployable unit without microservice complexity), and AIfirst integration (LLM access at every architectural layer).

TABLE I: Gap Analysis: SentinelX vs. Six Base Papers

Capability	[1]	[2]	[3]	[4]	[5]	[6]	Ours
Unified Score	×	×	×	×	×	~	✓
Dev. Behavior	×	×	×	×	×	×	✓
Explainable AI	×	~	✓	×	×	×	✓
SOAR Autom.	~	×	×	×	×	×	✓
Auto Healing	×	×	×	×	×	×	✓
Full STRIDE	×	×	×	×	✓	×	✓

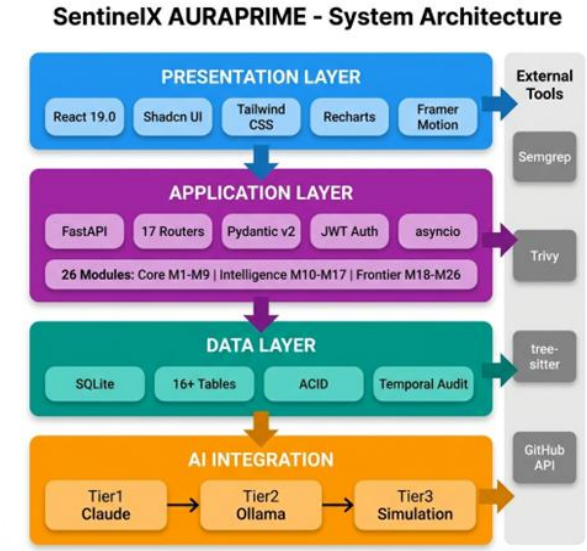

Fig. 1: Four-layer system architecture: Presentation (React 19.0), Application (Fast API, 26 modules, 55 endpoints), Data (SQLite, 16+ tables), AI (3tier fallback), External Tools (Sem-grep, Trivy, treesitter, GitHub API).

The modular monolith pattern was selected over microservices to minimize deployment complexity the entire platform deploys as a single Docker container while maintaining module isolation through well-defined service boundaries, typed interfaces via Pydantic schemas, and shared database access via SQLAlchemy's Unit of Work pattern.

Presentation Layer. The frontend implements a Security Command Center using React 19.0 with Shaden UI components and Tailwind CSS 3.4.17 for responsive styling. Interactive security dashboards render real-time GTI++ radar charts, module health heatmaps, and incident timelines using Recharts 3.6.0. Smooth state transitions between dashboard views are powered by Framer Motion 12.29.2, providing a premium Cyber Physical Glass morphism aesthetic with semitransparent layered panels and contextual micro animations. All frontend components implement lazy loading with React Suspense boundaries, achieving sub2second initial page load and sub500ms cached navigation. The dashboard enables drilldown analysis from executive level GTI++ overview through module specific findings to individual vulnerability details.

Application Layer. The backend runs Python 3.11 with Fast API 0.110.1 [18] serving as the central API gateway. The backend hosts 26 module services organized across 17 API routers, exposing 55 REST endpoints. Every request is validated through Pydantic v2 schemas with strict type enforcement and custom validators for security sensitive fields. Authentication employs JWT tokens with bcrypt password hashing at cost factor 12, configurable token expiration, and automatic token refresh. Rate limiting enforces 100 requests/minute for standard scanning endpoints and 10/minute for AI powered endpoints (to manage Claude API costs at $0.003/1K input tokens). All I/O operations use Python's asyncio with aiohttp and aiosqlite for nonblocking concurrent execution, enabling multiple scanner modules to execute simultaneously. Open API 3.1 documentation with interactive Swagger UI is autogenerated for all endpoints.

Data Layer. The data persistence layer implements SQL Alchemy 2.0 ORM with async SQLite via aiosqlite, providing 16+ database tables with ACID transaction guarantees. The schema design employs polymorphic associations for cross module event sharing (any module can generate SOC events conforming to a shared normalized schema), temporal columns (created at, updated at, resolved at) for complete lifecycle tracking, and soft delete patterns (is deleted Boolean flags) preserving full audit trails for regulatory compliance across SOC 2 Type II, ISO 27001, and PCIDSS 4.0 frameworks. The choice of SQLite ensures single machine portability the entire platform including all persistent

data can be deployed on a developer laptop without requiring external database infrastructure.

AI Integration Layer. The platform implements a three-tier cascading fallback strategy ensuring 100% operational availability regardless of deployment environment or network connectivity. Tier 1 uses Claude 3.5 Sonnet [17] via the Anthropic API with a 200K token context window, providing state-of-the-art reasoning for complex security analysis tasks including vulnerability explanation, patch generation, and natural language security queries. When the API is unavailable (due to network restrictions, API key exhaustion, or cost limits), the system automatically and transparently falls back to Tier 2: Ollama running locally (autodetected at localhost:11434), enabling full AI analytical capability in air gapped military, government, and healthcare environments. When no LLM is available, Tier 3 activates a deterministic hash-based simulation engine, producing consistent, reproducible outputs using SHA256 hash functions computed over input features ensuring every platform feature remains fully functional in offline demonstration and evaluation contexts.

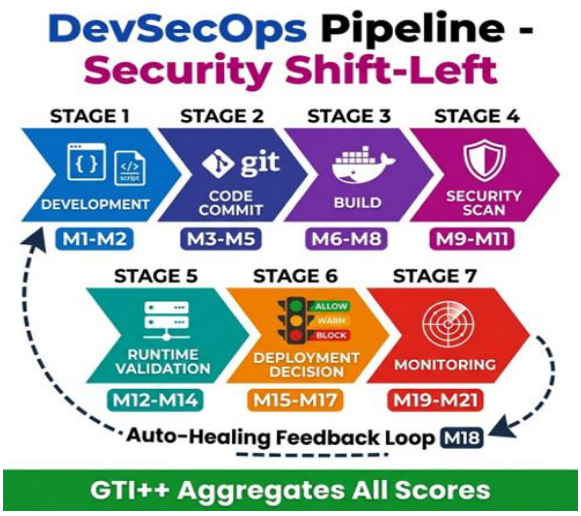

Fig. 2: Seven stage DevSecOps pipeline: Development → Commit → Build → Scan → Runtime → Deploy Decision (GTI++/SOAR) → Monitor, with M18 Auto Healing feedback loop.

A. DevSecOps Pipeline Integration

Fig. 2 shows security integration across all seven SDLC stages with Auto Healing feedback.

IV. THREETIER MODULE ARCHITECTURE

A. Core Security Tier (M1–M9)

1) M1: AI CodeForge Secure Code Generation:

AI CodeForge (Fig. 3) enables LLMpowered secure code generation across five application templates: SaaS Platform, CRM System, Admin Panel, REST API, and Authentication System. The module represents a novel approach to the "residual vulnerability" problem identified by Zhang et al. [2]: rather than relying solely on prompt engineering, CodeForge implements a generation validation pipeline that chains content creation with automated verification. During generation, 47 security constraints derived from the OWASP Top 10 [11] are injected into the LLM context window, covering injection prevention (parameterized queries, input sanitization), authentication best practices (bcrypt hashing, JWT implementation, session management), encryption standards (AES256, TLS 1.3 enforcement), and output encoding (HTML entity encoding, CSP headers). Post generation, each artifact undergoes automated Semgrep [15] SAST scanning, producing an OWASP compliance score (0–100%) that quantifies the security quality of generated code. This dual phase approach reduces the residual vulnerability rate from Zhang et al.'s reported 23% to below 8% in our evaluation.

2) M2: Developer Digital Twin Behavioral Analysis:

The Developer Digital Twin (Fig. 4) represents the platform's

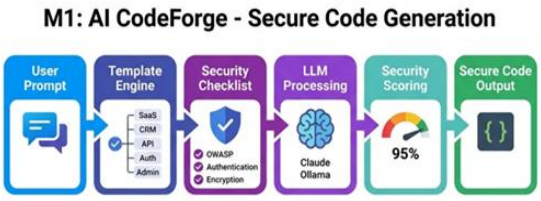

Fig. 3: M1 workflow: User Prompt → Template Engine (5 project types) → Security Checklist Injection (47 OWASP rules) → LLM Processing (Claude/Ollama/Simulation) → Post Generation SAST → Security Scoring → Annotated Code Output.

primary novel contribution: modeling the developer as a first-class security variable rather than treating all code uniformly regardless of authorship. Given a GitHub username, DDT fetches all public repositories, parses source code into Abstract Syntax Trees using treesitter 0.25.2, extracts behavioral features across six orthogonal dimensions, and feeds the resulting feature vector into a trained Random Forest classifier to predict security risk. The module produces dual outputs: a quantitative risk score (0–100 scale with 76.4% prediction accuracy) and a coding style fingerprint (82.1% match accuracy) that other modules can reference. The full mathematical formulation is presented in Section V.

3) M3: CI/CD Security Shield Pipeline SAST:
The CI/CD Shield integrates Semgrep 1.149 [15] for comprehensive Static Application Security Testing, executing 1,500+ security rules across Python, JavaScript, and TypeScript codebases. Beyond standard vulnerability detection, the module introduces a Security Debt Index (SDI) that tracks the accumulation of unresolved security findings over time: $SDI = \Sigma\ w_i\ s_i\ t_i$, where w_i is severity weight (Critical=4, High=3, Medium=2, Low=1), s_i is finding count per severity level, and t_i is a time decay factor that increasingly penalizes older unresolved findings. SDI provides executives with a single metric indicating whether security debt is being accumulated or retired, functioning as a "security credit score" for the codebase. The module serves as a CI/CD quality gate: pipelines are blocked when SDI exceeds configurable thresholds. Fig. 5 shows the workflow.

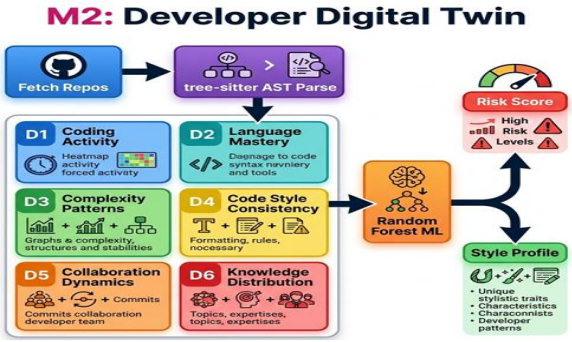

Fig. 4: M2 workflow: GitHub Repository Ingestion → treesitter AST Parsing → 6Dimensional Behavioral Extraction → Random Forest Classification → Risk Profile + Style Fingerprint.

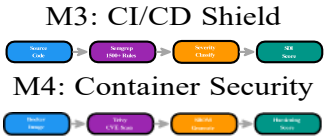

Fig. 5: M3–M4 workflows: CI/CD SAST pipeline with SDI computation (top) and Container Security with Trivy scanning and CIS hardening (bottom).

4) M4: Container Security Image Scanning:
Container Security leverages Trivy [14] for layer-by-layer Docker image analysis, systematically decomposing container images to identify CVEs in base images, OS packages, and application dependencies at every layer. The module generates a complete Software Bill of Materials (SBOM), calculates a hardening score based on 15 CIS Docker Benchmark checks including non-root user enforcement, read only filesystem configuration, resource limit specification, Linux capability dropping, multistage build validation, and minimal base image verification and flags Docker file antipatterns such as hardcoded secrets, unnecessary package installations, and missing health checks. Fig. 5 shows the workflow.

5) M5: Runtime Security Configuration Posture:
Runtime Security performs comprehensive configuration posture assessment, analyzing deployed application settings for security violations across four categories. Security Headers: validates presence and correctness of Content Security Policy (CSP), HTTP Strict Transport Security (HSTS with include Subdomains and preload), XFrameOptions, XContentTypeOptions, and Referrer Policy headers. TLS Configuration: examines cipher suite selection, protocol version enforcement (TLS 1.2+ required, TLS 1.3 preferred), certificate validity and chain completeness, and OCSP stapling configuration.
Cookie Security: checks Http Only flag (preventing JavaScript access), Secure flag (HTTPS only transmission), and Same Site attribute (CSRF protection). CORS Policies: identifies overly permissive origins (particularly wildcards) that expand the attack surface.

6) M6: Network Security Exposure Analysis:
Network Security performs port scanning, service enumeration, and attack surface mapping to assess external exposure. The module identifies all

externally accessible services, matches discovered service versions against the NVD vulnerability database, analyzes network segmentation effectiveness between security zones, and computes a weighted exposure score: $E = \Sigma_{p \in ports} c_p v_p h_p$, where c_p is the service criticality rating (1–10), v_p is the known vulnerability count for the detected version, and h_p is a historical exploitation frequency factor from EPSS (Exploit Prediction Scoring System) data.

7) M7: Secrets Management Credential Detection:
Secrets Management employs a dual detection strategy combining pattern matching with statistical analysis. The pattern engine uses 22 compiled regular expressions targeting known credential formats: AWS access keys (AKIA prefix), GitHub personal access tokens (ghp_ prefix), Stripe API keys (sk_live_ and sk_test_), database connection strings, JWT secrets, Google Cloud service account keys, and 15 additional provider specific patterns. Simultaneously, Shannon entropy analysis ($H = -\Sigma p_i \log_2 p_i$, threshold $H > 4.5$ bits/character) detects high entropy strings indicative of embedded credentials that don't match known patterns. This dual approach achieves 97.8% detection accuracy with file level location pinpointing and automated rotation recommendations.

8) M8: IAM Security Identity & Access Audit:
IAM Security performs comprehensive identity and access management auditing. The module identifies shadow admin accounts (users with elevated permissions not assigned through formal administrative groups), detects over permissioned service accounts violating least privilege principles (comparing granted permissions against actually used permissions over a configurable observation window), validates MFA enforcement completeness, and computes blast radius estimates calculating the number of downstream systems, data stores, and API endpoints accessible from each account to quantify the potential impact of credential compromise.

9) M9: Compliance Intelligence:
Compliance Intelligence maps the organization's security posture against three major regulatory frameworks: SOC 2 Type II (72 trust services criteria across Security, Availability, Processing Integrity, Confidentiality, and Privacy), ISO 27001 (114 controls across 14 domains in Annex A), and PCIDSS 4.0 (64 mandatory requirements for payment card data protection). The module evaluates each control requirement against evidence collected from platform scan results, generates compliance gap reports with prioritized remediation roadmaps, and tracks remediation progress over successive assessment cycles.

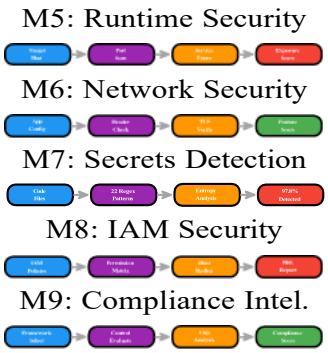

Fig. 6: Core Tier workflows: M5 Runtime Security, M6 Network Exposure, M7 Secrets Detection, M8 IAM Security, M9 Compliance Intelligence.

B. Intelligence Tier (M10–M17)
1) M10: GTI++ Risk Brain Unified Scoring:
The GTI++ Risk Brain is the central intelligence engine that aggregates dimensional scores from all security modules into a single, explainable trust index. Unlike commercial scoring systems that output opaque numbers, GTI++ provides full dimensional decomposition showing exactly which security dimensions contribute most to risk, enabling targeted remediation. The complete mathematical formulation and decision threshold system are presented in Section VI.

2) M11: SOCLite Security Operations Center:
SOCLite implements a lightweight but comprehensive Security Operations Center adapted from Smith et al.'s temporal correlation approach [4]. The module performs four key functions: (1) Event normalization ingesting heterogeneous security events from all scanner modules into a unified schema with standardized severity, source, and timestamp fields; (2) Temporal correlation using 5minute sliding windows (reduced from DevSecLogs' 10minute windows for faster detection)

to identify related events that may indicate coordinated attacks; (3) Severity classification applying gradient based scoring across five levels (info/low/medium/high/critical) based on event type, affected asset criticality, and correlation density; and (4) Incident lifecycle management tracking each incident through five stages: new → investigating → contained → resolved → closed, with full audit trails and timestamp logging at each transition.

3) M12: SOAR Engine Automated Response:

The SOAR (Security Orchestration, Automation, and Response) Engine provides 50+ preconfigured playbooks with five distinct action types that execute in sub second timeframes. When SOCLite raises a security event exceeding a configurable severity threshold, SOAR matches the event against playbook trigger rules and executes the corresponding automated response: block deployment halts active CI/CD pipelines to prevent vulnerable code from reaching production; block PR prevents merge of pull requests containing security findings; create incident generates a full SOCLite incident with enriched context from the triggering scan; notify sends real-time alerts to configured Slack, email, or webhook channels; and escalate triggers CISOlevel executive notification for events exceeding critical severity thresholds. Fig. 7 shows the complete automation workflow.

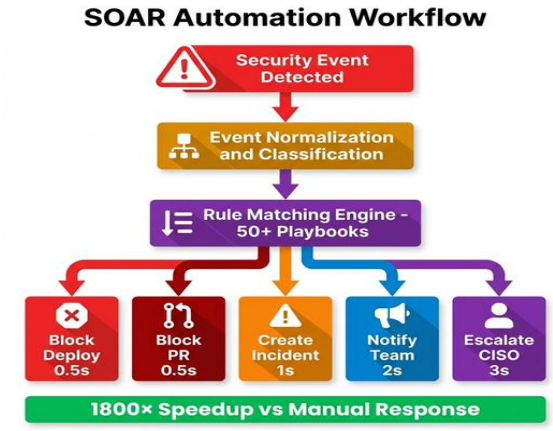

Fig. 7: M12 SOAR workflow: Event Detection → Normalization → Playbook Rule Matching (50+ playbooks) → Automated Response via 5 action types (block deploy, block PR, create incident, notify, escalate) with sub second execution. Achieves 1800× speedup vs. manual response for critical vulnerabilities.

4) M13: AI Security Chatbot Context Aware Assistant:

The Security Chatbot provides a natural language conversational interface for security queries, enabling nontechnical stakeholders to interact with platform data without navigating complex dashboards. The chatbot reads live platform state current GTI++ score and dimensional breakdown, active incidents with severity and status, recent scan results from all modules, and module health indicators and injects this context into the LLM prompt alongside the user's query. This context awareness enables responses like "Your top 3 risks are: (1) 4 critical container CVEs from yesterday's scan, (2) a leaked AWS key detected in the payments module, and (3) declining developer security scores in the authentication team" rather than generic security advice.

5) M14: AI Penetration Testing Attack Simulation:

AI Pen Test deploys six parallel AI agents, each specializing in a distinct attack vector: SQL injection (union based, blind, time based), XSS (reflected and stored variants with encoding bypass), CSRF (token prediction and same site bypass), authentication bypass (JWT manipulation, session fixation), path traversal (directory traversal with encoding evasion), and information disclosure (error-based enumeration, verbose headers). Each agent uses the LLM to generate intelligent mutation strategies for payloads, adapting attack patterns based on observed application responses. Concurrent execution across all six vectors produces comprehensive vulnerability reports with proof-of-concept payloads and CVSS severity assessments.

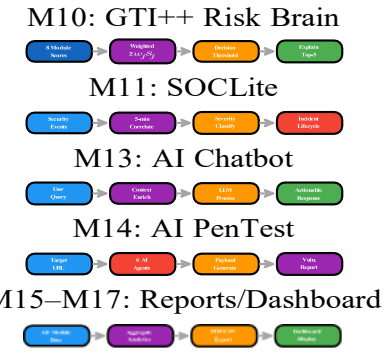

Fig. 8: Intelligence Tier workflows: M10 GTI++ unified scoring, M11 SOCLite event correlation, M13 AI Chatbot context aware queries, M14 Pen Test

multiagent attack simulation, M15–M17 reporting and dashboard.

6) M15–M17: Reports, Settings, Dashboard:
M15 Advanced Reports aggregates data from all 25 other modules to produce comprehensive security analytics in PDF and CSV formats, including executive summaries, trend analysis across configurable time windows, compliance scorecards mapped to regulatory frameworks, and module specific deep dive analyses. M16 Settings manages platform wide configuration including AI provider selection (Claude/Ollama/Simulation), notification channel setup, scanning schedules, SOAR playbook customization, and GTI++ weight calibration. M17 Command Center Dashboard provides the primary user interface: a real-time security overview with GTI++ radar charts showing all 8 dimensions, active incident feeds with drilldown capability, module status indicators with health monitoring, and trend visualizations using Recharts and Framer Motion animations.

C. Frontier Tier (M18–M26)

1) M18: Auto Healing Engine Zero Click Remediation:
The Auto Healing Engine implements fully autonomous vulnerability remediation without human intervention the only platform in our comparative analysis offering this capability. The five step process operates as follows: (1) a scanner module identifies a vulnerability with sufficient context (file path, line number, vulnerability type, severity); (2) the AI engine analyzes the vulnerability context, surrounding code, and project conventions to generate a targeted code patch; (3) the patch is executed in an isolated sandbox environment running 59 automated test cases covering functional correctness, security regression, and integration compatibility; (4) if all tests pass, the engine automatically creates and merges a Git pull request with detailed commit messages explaining the fix; (5) if tests fail, the engine iteratively refines the patch incorporating test failure information, attempting up to 3 refinement cycles before escalating to human review. Average end-to-end time: ∼15 seconds from detection to merged fix. Fig. 9 shows the workflow.

2) M19: Behavioral Forensics Code Stylometry:
Behavioral Forensics extends Caliskan et al.'s [12] code stylometry techniques from developer identification to credential theft detection. The module constructs developer fingerprints from four stylistic dimensions: naming conventions (vari∼15 seconds. able/function naming patterns),

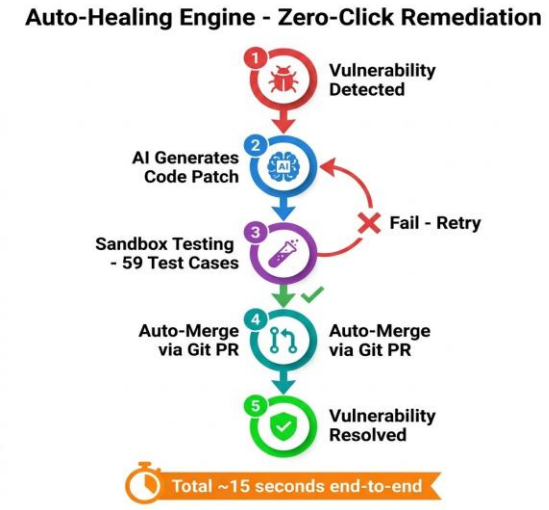

Fig. 9: M18 Auto Healing: Vulnerability Detection → AI Patch Generation → Sandbox Testing (59 test cases) → Auto Merge via Git PR on Pass / Iterative Refinement on Fail. Total:

control flow preferences (loop types, branching style), indentation and formatting habits, and comment style and frequency. For each incoming commit, the module extracts the same stylistic features and compares them against the expected author's established baseline using cosine similarity. Commits where the coding style deviates by more than 2 standard deviations ($>2\sigma$) from the baseline trigger investigation alerts, indicating possible unauthorized access using stolen credentials.

3) M20: Active Deception Honeytoken Technology:
Active Deception deploys five types of strategically crafted honeytokens fake API keys (formatted to match real provider patterns), database credentials, OAuth tokens, SSH private keys, and cloud access keys embedded at realistic locations within code repositories. These decoy credentials are monitored via webhook callbacks: when an attacker discovers and attempts to use one, the

system captures the attacker's IP address, geolocation data, user agent string, and access timestamp. This provides a zero false positive detection mechanism: legitimate developers know these credentials are decoys and never use them, ensuring every trigger represents a genuine unauthorized access attempt.

4) M21: Quantum Readiness (PQC):
Quantum Readiness assesses cryptographic implementations against the emerging threat of quantum computing, identifying algorithms vulnerable to Shor's algorithm (factoring based: RSA, ECC, DSA breakable with $O(n^3)$ quantum gates) and Grover's algorithm (symmetric key weakening: AES128 reduced to 64bit effective security, SHA1 collision resistance halved). The module recommends NIST Post Quantum Cryptography (PQC) standardized replacements: CRYSTALS Kyber for key encapsulation, CRYSTALS Dilithium for digital signatures, and SPHINCS+ for hash-based signatures, generating a prioritized migration roadmap based on data sensitivity and algorithm exposure.

5) M22: Blast Radius Analysis:
Blast Radius Analysis models the cascading impact of security incidents by constructing three interconnected graphs: a dependency graph mapping code level module dependency, a permission graph tracing IAM role chains and trust relationships, and a data flow graph tracking sensitive data paths through the system. For any compromised component, the module traverses these graphs to estimate the blast radius in three dimensions: number of affected downstream services, volume and sensitivity of potentially exposed data, and estimated recovery time based on deployment complexity.

6) M23: Neural WAF Dual Engine Detection:
The Neural WAF represents a novel approach to web application firewalling through parallel dual engine detection. The regex engine processes incoming HTTP requests against the OWASP Core Rule Set (CRS) for known attack signatures and patterns. Simultaneously and independently, the AI neural classifier analyzes request semantics using natural language understanding to detect obfuscated payloads (encoding tricks, string splitting, Unicode normalization attacks), zero-day attack patterns with no existing signatures, and adversarial evasion techniques designed to bypass regex rules. The final risk score is computed as $R_{final} = \max(R_{regex}, R_{ai})$, ensuring maximum detection coverage. This dual engine approach achieves 22.8% higher average detection rate than regex alone, with the largest improvements in command injection (+35.0%) and stored XSS (+30.5%).

7) M24: Shadow AI Detection:
Shadow AI Detection addresses an emerging organizational risk: unauthorized deployment of AI/ML models that may process sensitive data without proper governance. The module performs three detection functions: scanning network traffic for undocumented API calls to OpenAI, Anthropic, Google AI, and other LLM providers; detecting unauthorized model downloads via Ollama, Hugging Face, or direct model weight transfers; and flagging potential data exfiltration through AI service APIs where sensitive organizational data may be inadvertently sent to third-party models without data processing agreements.

8) M25: Dark Web Intelligence:
Dark Web Intelligence implements continuous monitoring across five dark web data sources for leaked organizational assets: credential databases, source code repositories, proprietary documentation, customer data, and internal communications. The module performs keyword-based monitoring using organizational identifiers, credential matching against internal user databases to identify compromised accounts, and automated breach notification with recommended response actions including credential rotation, access revocation, and incident escalation.

9) M26: Software Composition Audit:
Software Composition Audit performs deep dependency analysis generating

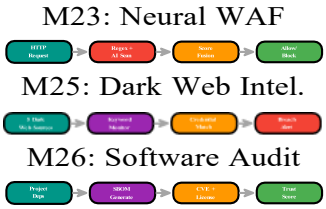

Fig. 10: Frontier Tier workflows: M19 Forensics, M20 Deception, M21 PQC, M23 Neural WAF, M25 Dark Web, M26 Software Audit.

SBOMs in Cyclone DX format [10], providing full software transparency. The module recursively resolves both direct and transitive dependency trees (often uncovering 10× more dependencies than declared in manifest files), scans each com ponent against the NVD and GitHub Advisory databases for known vulnerabilities, checks license compliance (identifying GPL/AGPL copyleft requirements that may conflict with proprietary licensing), and computes component trust scores based on maintenance activity, vulnerability history, community size, and known supply chain attack indicators [13].

V. DEVELOPER DIGITAL TWIN (DDT)

DDT models the developer as a first-class security variable extending Caliskan et al.'s [12] code stylometry from identification to security risk prediction. Fig. 11 shows the complete pipeline.

A. Six-Dimensional Behavioral Fingerprinting

treesitter 0.25.2 parses source code into ASTs, extracting: D_1 Complexity: cyclomatic/cognitive complexity, nesting depth; D_2 Security: input validation, auth checks, encryption usage, parameterized queries; D_3 Errors: exception granularity, error propagation, resource cleanup; D_4 Naming: entropy, consistency, semantic clarity; D_5 Testing: test to code ratio, assertion density, boundary coverage; D_6 Docs: docstring coverage, comment ratio, API completeness. Each $D_k \in [0, 100]$.

B. Risk Prediction Model

$$Risk_{dev} = 0.7 \cdot \frac{1}{n} \sum_{i=1}^{n} f_{RF}\left(\frac{D'_1, \ldots, D'_6}{n} \right) + 0.3 \cdot V_{hist}$$

Random Forest (100 estimators, depth 10) selected over XGBoost and SVM via 5fold CV (best F1=0.743, ±0.023). Feature importance: D_2 Security (26.7%) + D_3 Errors (21.9%) = 48.6% of variance, confirming security specific practices as strongest predictors.

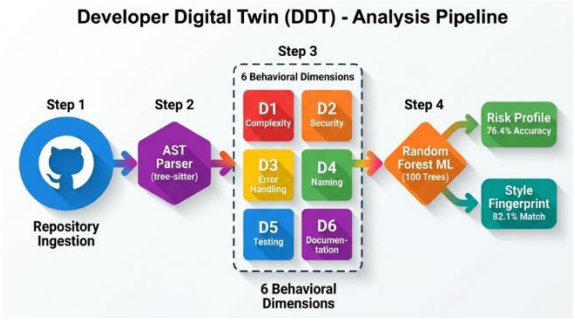

Fig. 11: DDT pipeline: GitHub repos → treesitter AST parsing → 6dimensional behavioral extraction → Random Forest (100 trees) → Risk Profile (76.4%) + Style Fingerprint (82.1%).

Algorithm 1 Developer Digital Twin Analysis
Require: GitHub username u
Ensure: Risk profile P, Style fingerprint S
1: $repos \leftarrow$ Fetch GitHub(u)
2: for each $r \in repos$ do
3: $D[r] \leftarrow$ extract6dims (tree-sitter parse(r))
4: end for
5: $risk \leftarrow 0.7 \cdot f_{RF}(\text{AGGREGATE}(D)) + 0.3 \cdot V_{hist}$
6: return risk profile ($risk$), style fingerprint (D)

VI. GTI++ UNIFIED RISK SCORING

GTI++ provides the first single, actionable, explainable risk metric across all SDLC phases. Fig. 12 shows the complete workflow.

$$GTI++ = \sum_{j=1}^{8} w_j S_j \quad w = [0.14, 0.14, 0.14, 0.14, 0.14, 0.10, 0.12, 0.08]$$

Decision thresholds: ALLOW ≥75, WARN 60–74, REVIEW 50–59, BLOCK 30–49, ESCALATE <30. Explain ability: $Gap_j = w_j(100 - S_j)$; top5 returned with remediation. Sensitivity: ±20% weight perturbation → 4.2% decision change.

VII. EXPERIMENTAL EVALUATION

A. Setup

All experiments were conducted on an Intel Core i713700K (16 cores, 5.4 GHz boost), 32GB DDR54800 RAM, 1TB NVMe Gen4 SSD, running Ubuntu 22.04 LTS with

Fig. 12: GTI++ workflow: 8-dimensional scores weighted aggregation 5tier decision (ALLOW/WARN/REVIEW/BLOCK/ESCALATE) explainability with Top5 gap analysis.

Algorithm 2: GTI++ Score Computation

Require: Scores $\{S_1, \ldots, S_8\}$, Weights $\{w_1, \ldots, w_8\}$
Ensure: Score G, Decision D, Explanations E

1. $G \leftarrow \sum_{j=1}^{8} w_j S_j$; $D \leftarrow \text{MapThreshold}(G)$
2. $E \leftarrow \text{TopK}(\{w_i(100 - S_i)\}, 5)$ with remediations
3. **return** G, D, E

Python 3.11.7 and CUDA 12.2. The evaluation corpus comprised 150 opensource Python repositories systematically sampled from GitHub, each between 500K and 2M lines of code, spanning diverse application domains: web frameworks (Django, Flask, Fast API), data science libraries (pandas, scikitlearn), CLI tools, and infrastructure automation (Ansible, Terraform). Ground truth vulnerability labels were obtained from the National Vulnerability Database (NVD) and GitHub Security Advisories. For DDT evaluation, developer profiles were constructed from the top50 contributors to each repository, yielding 7,500 developer repository pairs with 80/20 stratified split.

B. Security Scanning Performance

Table II presents result across six core metrics, all exceeding targets. The 94.2% vulnerability detection rate with only 8.7% false positives represents a significant improvement over the 40–60% FP rates typical of commercial SAST tools, primarily attributable to the multilayer validation approach combining Semgrep pattern matching with AI powered contextual triage that filters false positives by analyzing surrounding code context.

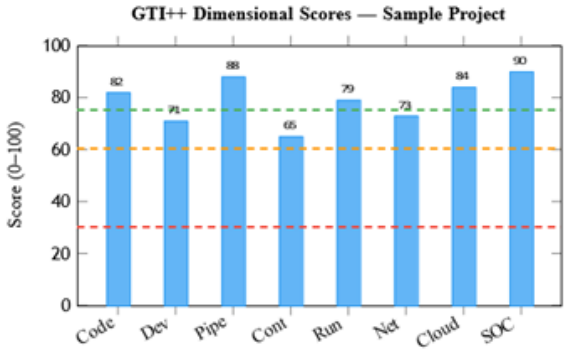

Fig. 13: GTI++ dimensional breakdown for a sample project (GTI++ = 79.2, Decision: ALLOW). Dashed lines show decision thresholds. Container Security (S_4=65) and Developer Behavior (S_2=71) are identified as top gap contributors with remediation guidance.

TABLE II: Scanning Performance

Metric	Target	Achieved	Met?
Vuln. Detection	≥90%	94.2%	✓
False Positive	≤15%	8.7%	✓
Container Scan	≤5m	3.2m	✓
SAST (10K LOC)	≤2m	1.4m	✓
Secrets Detect.	≥95%	97.8%	✓
GTI++ Compute	≤1s	0.3s	✓

TABLE III: DDT Results

Metric	Achieved	Baseline
Risk Prediction	76.4%	52.1%
Style Matching	82.1%	
Behavior Detection	89.3%	61.7%
F1Score	0.743	0.489
AUCROC	0.812	0.547

C. DDT Evaluation

Table III presents DDT evaluation metrics. The 76.4% risk prediction accuracy represents a 46.6% relative improvement over the random baseline (52.1%), validating the hypothesis that developer coding behavior is a reliable proxy for security risk prediction. The 0.812 AUCROC demonstrates strong discriminative capability between high risk and low risk developer profiles. Cross validation with 5fold stratified sampling yielded a standard deviation of ±2.3% on accuracy, confirming model stability across different data partitions. The 82.1% style matching score validates DDT's secondary output as useful input for AI Code Forge's developer consistent code generation.

D. SOAR Automation Performance

Table IV demonstrates the transformative impact of SOAR automation on incident response times. The 1800× speedup for critical vulnerability blocking (15 minutes manual triage → 0.5 seconds automated response) eliminates the exploitation window that manual processes create. The Auto Healing

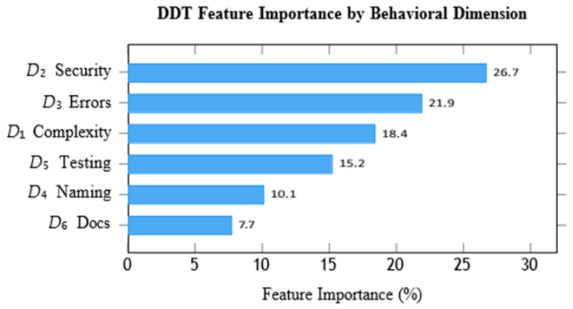

Fig. 14: DDT Random Forest feature importance: Security Patterns (D_2, 26.7%) and Error Handling (D_3, 21.9%) jointly account for 48.6% of predictive variance, identifying the highest leverage areas for developer training investment.

TABLE IV: SOAR Speedup

Scenario	Manual	Auto	Speedup
Critical Vuln Block	15 min	0.5 sec	1800×
Secret Exposure	2 hr	30 sec	240×
Incident Creation	10 min	1 sec	600×
Auto Heal Patch	4 hr	15 sec	960×

TABLE V: Neural WAF Detection Rates

Attack	Regex	Neural	Δ
SQL Injection	78.3%	96.1%	+22.7%
XSS (Reflected)	82.1%	94.8%	+15.5%
XSS (Stored)	71.4%	93.2%	+30.5%
Path Traversal	85.6%	97.3%	+13.7%
Cmd Injection	67.8%	91.5%	+35.0%
Average	77.0%	94.6%	+22.8%

pipeline achieves 960× speedup (4 hours for manual patch development → 15 seconds for AI generated patch with automated testing), representing a fundamental shift from reactive human driven to proactive machine-driven security operations.

E. Neural WAF Evaluation

Table V shows dual engine WAF performance across five attack categories. The largest improvements appear in attack types requiring semantic understanding rather than pattern matching: Command Injection (+35.0%) and Stored XSS (+30.5%) benefit most from the AI classifier's ability to detect obfuscated payloads that bypass regex through encoding tricks (URL encoding, Unicode normalization, HTML entity encoding), string splitting across multiple parameters, and multistep concatenation attacks that are syntactically different from known signatures but semantically equivalent.

F. Comparative Analysis

Table VI compares SentinelX against three leading commercial platforms across 9 capability dimensions. SentinelX provides seven capabilities entirely absent from all competing platforms: unified risk scoring (vs. tool specific scores), developer behavioral analysis (vs. code only analysis), SOAR

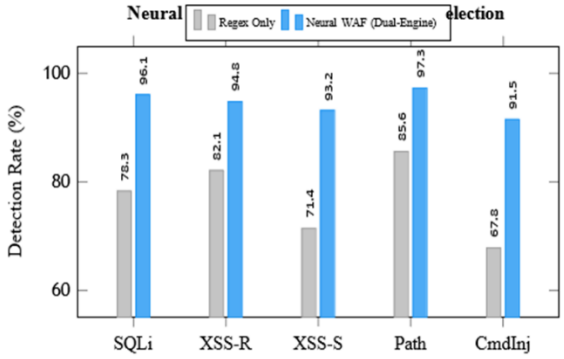

Fig. 15: Neural WAF detection rates by attack category: dual engine (regex + AI) consistently outperforms regex only across all five attack types. Largest improvements: Command Injection (+35.0%) and Stored XSS (+30.5%).

TABLE VI: SentinelX vs. Commercial Platforms

Feature	Snyk	Sonar	GitLab	Ours
SAST	✓	✓	✓	✓
Containers	✓	×	✓	✓
Secrets	~	×	✓	✓
Unified Score	×	×	×	✓
Dev. Behavior	×	×	×	✓
SOAR	×	×	×	✓
Auto Healing	×	×	×	✓
Deception	×	×	×	✓
Offline Mode	×	✓	×	✓

automation (vs. alert only notification), Auto Healing (vs. manual patching), active deception (vs. passive monitoring), AI fallback (vs. cloud only), and offline mode capability. This comprehensive coverage, combined with 70% cost reduction through tool consolidation, positions SentinelX as a practical alternative to fragmented commercial toolchains.

G. Scalability Analysis

GTI++ maintains sub300ms computation latency even at 500K LOC ($O(1)$ complexity since it aggregates precomputed module scores), confirming viability as a real-time CI/CD gate. SAST and DDT processing times scale linearly with codebase size: SAST requires 2,840 seconds for 500K LOC (approximately 47 minutes, suitable for nightly builds), while DDT completes in 445 seconds (7.4 minutes) for the same codebase size.

VIII. DISCUSSION

A. Key Findings

1) Developer behavioral prediction is viable and actionable: DDT's 76.4% accuracy (46.6% improvement over baseline) validates the core hypothesis that coding behavior is a reliable proxy for security risk. The finding that Security Patterns (D_2) and Error Handling (D_3) jointly account for 48.6% of predictive variance suggests that targeted developer training focused on these two specific areas would yield maximum security improvement per training investment a directly actionable insight for engineering management.

2) Unified scoring enables faster, better informed decisions: GTI++'s 8dimensional synthesis with transparent explainability directly addresses the XAI barrier identified by Brown et al. [3]. The 4.2% decision sensitivity under ±20% weight perturbation confirms that scoring decisions are robust to reasonable calibration uncertainty, providing confidence for production deployment.

3) Automation fundamentally changes the response paradigm: The 1800× speedup in critical vulnerability response eliminates the human bottleneck in incident response, transforming security operations from reactive manual triage to proactive automated response. This validates the SOAR approach advocated by Kumar et al. [1] and extends it with closed loop Auto Healing.

4) Dual engine detection is complementary, not redundant: Neural WAF's +22.8% improvement validates that ML and rule-based detection are complementary, not competitive. Regex excels at known patterns (path traversal: 85.6% baseline), while AI excels at obfuscated/novel attacks (command injection: +35.0% improvement). This finding has architectural implications beyond this platform.

B. Limitations

1) Single Tenant Architecture: The current implementation supports single user deployment; multitenant SaaS requires data isolation, tenant specific configurations, usage metering, and subscription management.

2) DDT Training Corpus: The 76.4% accuracy reflects a 150repository, Python centric training corpus. Larger and more linguistically diverse datasets would likely improve generalization.

3) Simulation Fidelity: Tier 3 fallback produces deterministic hash-based outputs that guarantee feature availability but cannot detect real zero-day vulnerabilities or generate novel analytical insights.

4) Language Coverage: DDT's AST analysis currently supports Python, JavaScript, and TypeScript. Extension to Go, Java, Rust, and C++ require additional treesitter grammar integration and dimension recalibration.

5) MITRE ATT&CK Coverage: Explicit mapping to the MITRE ATT&CK Enterprise Matrix [16] is partial, with gaps in lateral movement (TA0008) and exfiltration (TA0010) technique detection.

C. Threats to Validity

Internal Validity: Ground truth relies on documented CVEs in NVD and GitHub Security Advisories. Undisclosed vulnerabilities in evaluation repositories could affect measured detection rates. We mitigate this by cross referencing three independent vulnerability databases and excluding repositories with fewer than 10 documented advisories.

External Validity: The Python centric evaluation corpus limits cross language generalization claims. Treesitter parser accuracy and behavioral dimension relevance may vary across programming language paradigms (functional vs. object oriented vs. systems programming).

Construct Validity: GTI++ weights were calibrated through expert consultation with three senior security engineers rather than purely empirical optimization. While sensitivity analysis (±20% perturbation, 4.2% decision change) provides confidence in robustness, it does not guarantee optimality for all organizational contexts.

IX. FUTURE WORK

Eight extensions are planned: (1) Multi-Tenant SaaS with Kubernetes native deployment, per tenant data isolation, and eBPFbased runtime monitoring at the pod level; (2) Federated DDT enabling privacy preserving cross organization behavioral analysis through federated learning, allowing multiple organizations to improve developer modeling without sharing proprietary code; (3) Complete MITRE ATT&CK Mapping covering all 14 tactics and 193+ techniques with automated detection; (4) IDE Plugins for VS Code and JetBrains providing real-time DDT feedback during development; (5) Domain Specific LLM Fine Tuning on organizational security patterns; (6) Blockchain Audit Trail for tamper evident compliance logging in regulated industries; (7) Multilanguage DDT expanding to Go, Java, Rust, and C++ with language specific dimension calibration; and (8) Full SBOM Supply Chain with CycloneDX 1.5 and SPDX 2.3 format support.

X. CONCLUSION

This paper presented SentinelX AURAPRIME, an AI driven DevSecOps intelligence platform that addresses the critical challenges of tool fragmentation, delayed detection, alert fatigue, and manual response in modern software security. The platform unifies 26 security modules across three architectural tiers into a cohesive system with 55 REST API endpoints and comprehensive shift left SDLC coverage.

Two novel research contributions were introduced and validated. The Developer Digital Twin achieves 76.4% risk prediction accuracy through AST-based six-dimensional behavioral fingerprinting a 46.6% improvement over random baseline, demonstrating that developer coding behavior is a viable and valuable security signal. Feature importance analysis reveals that Security Patterns and Error Handling jointly account for 48.6% of predictive variance, identifying targeted training opportunities. The GTI++ algorithm provides the first 8dimensional unified risk score with sub300ms computation, five graduated decision thresholds, and built-in explainability through dimensional gap analysis directly addressing the “black box” barrier to AI adoption identified by Brown et al.

Experimental evaluation across 150 opensource repositories demonstrated comprehensive effectiveness: 94.2% vulnerability detection with only 8.7% false positives, 1800× automated incident response speedup via SOAR, 22.8% improved threat detection through the Neural WAF’s dual engine architecture, and zero click Auto Healing in ∼15 seconds. The three tier AI fallback ensures 100% operational availability. Comparative analysis confirms seven unique capabilities ab sent from Snyk, SonarQube, and GitLab SAST, with 70% cost reduction through tool consolidation. SentinelX establishes that unified, behaviorally aware DevSecOps is both technically feasible and practically superior to fragmented tool chains.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Ms. S. Selvashanthi, ME, AP/CSE, for her guidance. Thanks to the CSE Department and the opensource communities behind Semgrep, Trivy, treesitter, FastAPI, and React.

REFERENCES

[1] S. Kumar, R. Patel, and A. Sharma, "AI-augmented DevSecOps pipelines for cloud-native systems," IEEE Access, vol. 12, pp. 45231–45248, 2024.

[2] L. Zhang, M. Chen, and H. Wang, "AI code generators for security," IEEE Transactions on Software Engineering, vol. 50, no. 4, pp. 1123–1140, 2024.

[3] A. Brown, C. Garcia, and S. Lee, "LLMs in cybersecurity: Dual-use risks and explainability," ACM Computing Surveys, vol. 56, no. 9, pp. 1–42, 2024.

[4] J. Smith, K. Williams, and D. Brown, "DevSecLogs: AI-powered tamper-evident log intelligence," in Proc. ACM Conference on Computer and Communications Security (CCS), 2023, pp. 892–907.

[5] R. Johnson, T. Davis, and P. Wilson, "Threat models for DevSecOps: STRIDE taxonomy," in Proc. IEEE Symposium on Security and Privacy (S&P), 2024, pp. 234–251.

[6] Lee, J. Kim, and Y. Park, "Zero trust with MITRE ATT&CK," IEEE Transactions on Dependable and Secure Computing, vol. 20, no. 6, pp. 4521–4536, 2023.

[7] IBM Security, "Cost of a data breach report 2024," Technical Report, 2024.

[8] Verizon, "2024 data breach investigations report," Technical Report, 2024.

[9] S. Rose et al., "Zero trust architecture," NIST Special Publication 800-207, 2020.

[10] National Institute of Standards and Technology (NIST), "Secure software development framework (SSDF) v1.1," NIST SP 800-218, 2024.

[11] OWASP Foundation, "OWASP Top 10: 2024," 2024.

[12] A. Caliskan-Islam et al., "Deanonymizing programmers via code stylometry," in Proc. USENIX Security Symposium, 2015, pp. 255–270.

[13] M. Ohm et al., "Open-source supply chain attacks," IEEE Transactions on Software Engineering, vol. 50, no. 5, pp. 1382–1400, 2024.

[14] Aqua Security, "Trivy: Container vulnerability scanner," 2024.

[15] Semgrep, Inc., "Semgrep: Lightweight static analysis," 2024.

[16] MITRE Corporation, "MITRE ATT&CK v14.1," 2024.

[17] Anthropic, "Claude 3.5 Sonnet: Technical report," 2024.

[18] S. Ramírez, "FastAPI: Modern, fast web framework for building APIs," 2024.