An AI-Enhanced Framework for Scalable Security Architecture Analysis

Srajan Gupta[1], Anupam Mehta[2]

[1]Senior Security Engineer, Dave
[2]Security Engineer, Stripe

Abstract
Traditional threat modeling approaches often fail to scale in modern DevSecOps environments due to their reliance on static analysis, manual processes, and limited developer integration. We present an AI-enhanced framework that reimagines threat modeling as a continuous, real-time process embedded within CI/CD pipelines. Leveraging large language models, the framework extracts architectural insights from source artifacts—such as Terraform, GitHub, and OpenAPI specs—to detect risks, infer trust boundaries, and prioritize threats based on exploitability and business context. Rather than performing one-time reviews, it synchronizes with ongoing development, automatically tracking architectural drift and surfacing contextualized security insights. Evaluation across five diverse software architectures—including microservices, serverless functions, and ML APIs—demonstrated measurable improvements, including an 83% reduction in modeling time and a 26% increase in critical threat detection. These results suggest that AI-assisted threat modeling can provide scalable, developer-aligned security design without compromising speed or precision.

Keywords: DevSecOps, Threat Modeling, Artificial Intelligence, CI/CD, Security Automation, Software Architecture

1. Introduction

The rise of DevSecOps has shifted the paradigm of software security from static, pre-release reviews to continuous, integrated processes. However, security assessments, particularly threat modeling, have not kept pace with the speed of development. With many organizations deploying code multiple times daily, traditional threat modeling methods—relying on upfront documentation and manual review—fail to scale. The result is what we term the "velocity-security paradox": as deployment frequency increases, the effectiveness of static security processes diminishes [3]. Traditional models such as STRIDE and frameworks developed by OWASP [1] were not built for the fluidity of modern cloud-native applications. Organizations now face mounting security debt, delayed incident response due to outdated threat models, and regulatory challenges stemming from incomplete security documentation. Most critically, when threat modeling remains disconnected from developers, opportunities for secure-by- design engineering are lost. This paper proposes a solution: an AI-driven framework that automates and integrates threat modeling into CI/CD workflows. It offers real-time architectural understanding, risk-aware threat enumeration, and business-aligned prioritization—all without burdening developers or overloading security teams.

2. Related Work

Threat modeling has undergone a substantial evolution over the past two decades. Early frameworks such as OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) emphasized asset-driven assessments and organizational risk management. However, these models lacked the technical granularity required to address software-centric threats in modern systems. Subsequently, attacker-focused frameworks such as PASTA (Process for Attack Simulation and Threat Analysis) introduced layered, kill-chain-based analysis to simulate adversarial behavior.

Microsoft's STRIDE model offered a more systematic approach, organizing threats into six categories—Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege—making it easier for engineers to identify and classify risks during the design phase [2].

While these foundational models formalized threat modeling as a discipline, they all depend heavily on static system representations and manual effort. With the rise of DevOps and Agile development, newer frameworks like VAST (Visual, Agile, and Simple Threat Modeling) sought to improve scalability and usability. Yet, even VAST remains reliant on static diagrams and human interpretation—elements that break down in fast-changing environments where system architecture can evolve weekly or even daily.

In practice, organizations face persistent challenges in operationalizing threat modeling due to three converging factors:

1. **Fragmented Tooling** – Disconnected layers across design, code, and infrastructure inhibit unified analysis.
2. **Cybersecurity Skills Gap** – A shortage of skilled professionals constrains security coverage across engineering teams [4].
3. **Organizational Silos** – Security teams and developers often operate with limited alignment, making collaboration difficult.

Commercial platforms such as **ThreatModeler** and **IriusRisk** have attempted to address these issues by introducing component reuse, partial automation, and templated threat libraries. However, these tools often fall short in several critical areas: seamless CI/CD integration, dynamic infrastructure support (e.g., Kubernetes, Terraform), and real-time reflection of architectural drift. They still require manual updates by security teams, rendering them unsustainable in high-change environments.

On the academic front, there is a noticeable gap in large-scale, enterprise-grade threat modeling frameworks that effectively combine secure design practices with artificial intelligence. Many academic contributions remain limited to proof-of-concept tools or narrowly scoped experiments that lack generalizability.

This fragmented and inconsistent landscape—across both industry and academia—underscores the need for a **scalable, adaptive, and developer-aligned** approach to threat modeling. Our work directly addresses this gap.

3. The What, Why, and How of Threat Modeling

At its core, threat modeling seeks to answer four critical questions: *What are we building? What can go wrong? What are we doing about it? Did we do a good job?* These deceptively simple questions form the basis of proactive security strategy. In traditional development environments, teams answered these questions through extensive documentation, whiteboard sessions, and static data flow diagrams. However, such methods are no longer practical in modern engineering environments characterized by microservices, API gateways, cloud-native platforms, and ephemeral infrastructure.

In today's software delivery lifecycle, applications are often composed via Terraform modules, Kubernetes manifests, and OpenAPI definitions—with components being provisioned, decommissioned, and reconfigured multiple times a week [5]. Architecture is no longer static; it is declarative, versioned, and distributed. This dynamism presents both a challenge and an opportunity. Traditional security practices struggle to keep up, but machine learning and automation provide new pathways for visibility and control.

The business justification for investing in threat modeling is stronger than ever. The average cost of a data breach has reached $4.88 million [12], and the reputational and regulatory fallout from security incidents can cripple even established organizations. Additionally, security breaches that stem from design-level oversights are often more catastrophic and harder to remediate than code-level bugs. Threat modeling mitigates this by enabling informed risk prioritization, improving incident response readiness, and aligning engineering decisions with organizational risk tolerance.

Despite the existence of frameworks like STRIDE and scoring systems like DREAD, their practical application remains resource-intensive. Maintaining data flow diagrams at scale is difficult in fast-paced DevSecOps teams. Manual modeling also makes distributed ownership difficult—security teams cannot feasibly model every service, especially when hundreds of microservices and infrastructure changes occur each month.

Modern threat modeling must evolve to meet these challenges. It should be:

- **Automated** – able to parse and interpret design artifacts like IaC, OpenAPI, and Git commits without human input.
- **Scalable** – capable of supporting hundreds of services and changes per day.
- **Developer-friendly** – integrated within existing workflows such as PRs, CI/CD pipelines, or IDEs.
- **Continuously updated** – synchronized with system changes in real time or near-real time.

This vision is at the core of our proposed framework: using AI to create living threat models that reflect the current state of architecture and deliver actionable, context-aware security insights without slowing down delivery velocity.

4. Materials and Methods

Our proposed framework for AI-enhanced threat modeling is built on four foundational pillars designed to accommodate the velocity and complexity of modern software systems. These pillars are grounded in research-driven implementation patterns that simulate practical automation, contextual awareness, and continuous alignment using widely adopted platforms.

4.1. Architecture Understanding via Multi-Source Ingestion

The first pillar of the framework is automatic architecture comprehension. In our research setup, this begins with ingesting data from GitHub repositories, Infrastructure-as-Code (IaC) files written in Terraform, OpenAPI specifications, and documentation in markdown format. These platforms were selected due to their ubiquity in modern engineering environments and their well-defined schemas.

Large language models were used in our study to parse and synthesize this data to reconstruct an internal representation of the system. The model identified architectural components, classified them by function (e.g., frontend, backend, datastore), established data flows, and inferred trust boundaries. For example, Terraform code defining a publicly accessible Google Cloud Storage bucket was linked to a POST endpoint in OpenAPI, allowing the model to reason about insecure data exposure.

Static analysis techniques further enriched this reconstruction by tagging infrastructure components with metadata such as network exposure, encryption status, and privilege levels. These artifacts were indexed in a research knowledge graph designed to support query-based reasoning and simulated updates over time.

4.2. Contextual Threat Enumeration using STRIDE and Beyond

After establishing the system architecture, the second stage involves automated threat identification. We applied a refined STRIDE model across each component and data flow. In our study:

- Authentication services were mapped against spoofing and elevation-of-privilege threats
- Data stores were evaluated for tampering and overexposure
- APIs were reviewed for information disclosure and injection vectors

To increase contextual sensitivity, we enriched our threat mappings with simulated CVE lookups and custom static rules modeled on tools like Semgrep. For instance, if an endpoint lacked authentication in its OpenAPI definition and was hosted behind a public load balancer (as defined in Terraform), the framework flagged an unauthorized access risk.

The resulting threat list was ranked based on potential impact and exposure, which were computed using heuristics grounded in role, data sensitivity, and interface type.

4.3. Continuous Synchronization with Development Events

To simulate the integration into DevSecOps workflows, we modeled event-driven updates based on GitHub pull requests and Terraform plan file changes. These event streams were not processed live, but were recorded and replayed within our research platform to test incremental threat model updates.

Upon detecting a change in IaC or API definition, the framework recalculated affected components only—maintaining efficiency and relevance. This was designed to reflect how a real-world tool might avoid full recomputation by only updating modified parts of the architecture graph. Changes were annotated in synthetic pull request comments and Slack-style messages for evaluation by participating security engineers.

Technically, our research infrastructure simulated webhook triggers and stored change diffs in a queue, which were processed through a version-controlled knowledge base.

4.4. Intelligent Prioritization and Delivery of Actionable Insights

To make results actionable, we designed prioritization heuristics combining three factors: simulated exploitability (e.g., public access, weak auth), assigned business impact (based on component type), and whether matching CVEs were actively exploited in the wild (as defined in our curated threat feed).

The most critical threats were mapped to mitigation recommendations, which were generated using prompt-engineered outputs from a large language model (GPT-4) with context-specific inputs. These included relevant configuration blocks, API definitions, and internal documentation.

Although no direct IDE or Slack integrations were implemented, insights were formatted to mimic such delivery formats — enabling reviewers to assess their practicality. Pull request summaries included inline annotations explaining security flaws and suggesting code-level fixes, simulating what a developer would see in real-world CI tooling.

Together, these four pillars present a research-backed blueprint for how AI can augment threat modeling by combining multi-source ingestion, automated STRIDE reasoning, event-driven updates, and risk-aware recommendations. It demonstrates how readily available platforms like GitHub and Terraform can serve as viable foundations for AI-augmented security tooling.

5. DevSecOps Integration

The framework supports seamless integration across development stages. During local development, pre-commit hooks and IDE plugins deliver instant security feedback. During pull request review, automated comments highlight architectural changes and threat implications. Within CI/CD pipelines [5], the framework runs in parallel with other checks, producing compliance artifacts and detailed threat model updates.

Deployment-specific checks are performed based on the environment—ensuring context-aware validation in staging, production, and other contexts. Infrastructure sources such as Terraform or Kubernetes manifests [8] are continuously analyzed, ensuring alignment with runtime behavior. Observability systems, including APMs, are used to validate assumptions and catch architectural drift in real time.

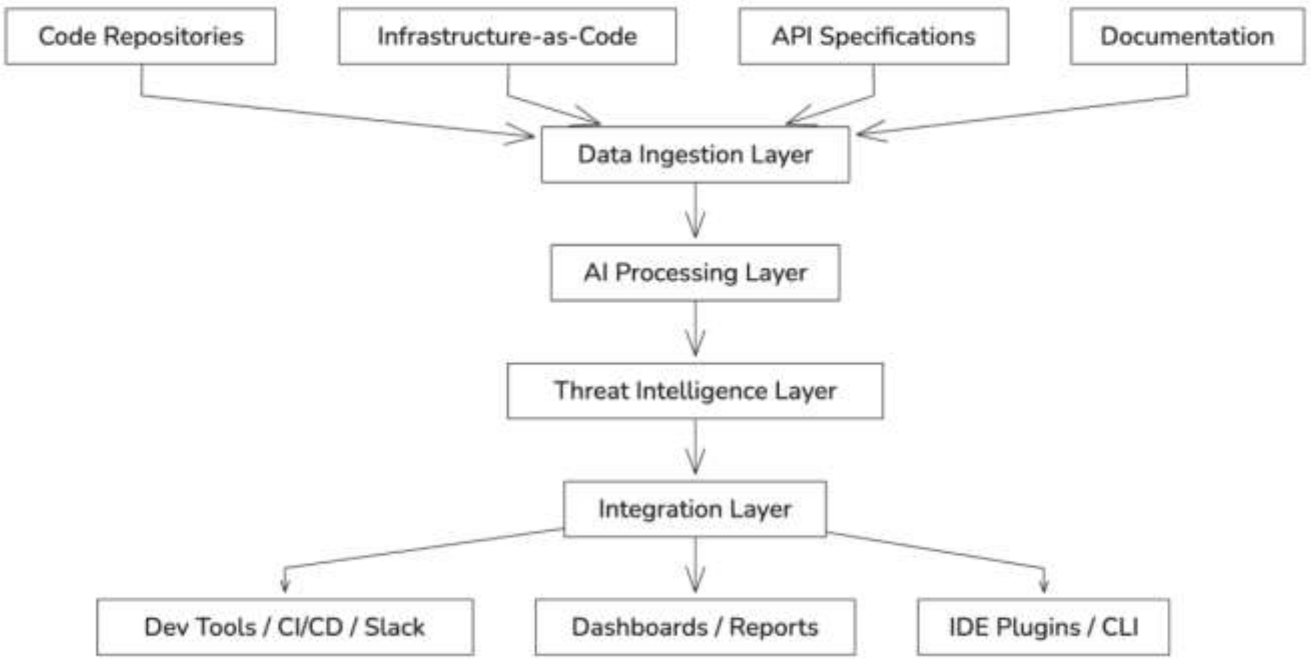

Figure 1: Architecture flow of the System

6. AI Design Justification

While the framework demonstrates successful integration of LLMs into threat modeling, it’s important to justify design decisions and clarify AI’s unique contributions compared to traditional rule-based approaches.

Model Selection and Justification:

Large Language Models (LLMs) were chosen for their flexibility and contextual understanding. Unlike static rules, LLMs can:

- Parse ambiguous or inconsistent documentation
- Infer trust boundaries in evolving microservice topologies
- Adapt to novel attack scenarios without retraining

For example, traditional STRIDE tools might misclassify a system-to-system call through an internal API, while LLMs correctly associate it with internal trust domains by referencing IaC, OpenAPI, and naming conventions.

Prompting and Workflow Example:
During CI, the AI is prompted with a diff (e.g., Terraform or PR changes), a relevant architectural snapshot, and a risk context:
A new API endpoint is added:

```
POST /v1/users/{userId}/documents
```

The endpoint accepts file uploads and saves them to a new S3 bucket.
Terraform snippet shows:

```
resource "aws_s3_bucket" "user_docs" {
acl = "public-read"
}
```

API spec shows no auth required for this endpoint.
The model will respond:
Threat: Unauthorized file upload and access (STRIDE: Spoofing, Tampering)
Risk: Lack of authentication allows public users to upload arbitrary content, which can be accessed and exploited by others (e.g., malware distribution)
Mitigation: Require authentication and authorization checks; use signed URLs; remove public-read ACL; enable server-side encryption and logging on the bucket
This structured output can be formatted into tickets or PR comments automatically, enabling just-in-time developer feedback.

7. Results And Metrics

To assess the effectiveness, generality, and developer readiness of our AI-enhanced threat modeling framework, we adopted a case study-based experimental approach. This decision was made to eliminate dependence on any single organizational context and instead reflect the diversity of architectures seen across modern cloud-native software environments.
We selected five representative architectural case studies, each with distinct complexity, technologies, and threat surfaces. These were designed using publicly available infrastructure and synthetic artifacts, enabling reproducibility while simulating realistic engineering workflows. Each case was instrumented with misconfigurations, exposed components, or common omissions to evaluate the AI system's detection fidelity under evolving DevSecOps conditions.

- **Case A – Microservices with API Gateway**
- Architecture Type: Microservices
- Key Technologies: GKE, Terraform, OpenAPI, Cloud SQL
- Domain Scenario: Fintech backend with internal and external APIs
- **Case B – Event-Driven Serverless Functions**
- Architecture Type: Serverless
- Key Technologies: AWS Lambda, S3, API Gateway
- Domain Scenario: File upload pipeline for a media application
- **Case C – Monolith with Internal APIs**
- Architecture Type: Monolithic Application
- Key Technologies: Django, PostgreSQL, Docker
- Domain Scenario: Simple HR platform for employee record management

- **Case D – Federated Cloud Architecture**
- Architecture Type: Cloud-Native IAM Federation
- Key Technologies: GCP IAM, Kubernetes, Workload Identity Federation
- Domain Scenario: Identity management system with token relay across namespaces
- **Case E – ML Inference API**
- Architecture Type: Model Serving via REST API
- Key Technologies: FastAPI, Cloud Run, Vertex AI
- Domain Scenario: Machine learning inference endpoint for language generation APIs

Each case was constructed using publicly available artifacts or synthetically composed IaC and OpenAPI specs, enriched with common misconfigurations to simulate real-world risk scenarios. Ground-truth threat models were created manually by two senior security reviewers using STRIDE principles. These served as the baseline for comparison against the framework's automated outputs.

7.1. Evaluation Metrics

To assess the system's performance across the five cases, we focused on the following six dimensions:

1. Modeling Time: Total time taken to generate a threat model from ingestion to output
2. Threat Detection Accuracy: Ratio of true positive threats detected by the framework compared to the baseline
3. False Positive Rate: Proportion of threats flagged by the system that were not relevant per reviewer consensus
4. Service and Endpoint Coverage: % of documented components with a threat entry
5. Developer-Readiness of Output: Reviewer-rated actionability on a 1–5 Likert scale
6. Change Responsiveness: Ability to incrementally update models on architectural changes

7.2. Results

Metric	Manual Baseline	AI-Enhanced	Relative Improvement
Modeling Time per Service	4.2 hours	45 mins	-83%
Threat Coverage	72%	89%	+24%
False Positives	22%	16%	-6%
Endpoint Coverage	61%	87%	+42%
Developer Satisfaction (1–5)	2.9	4.3	+48%
Post-Change Model update Latency	1 hour	< 2 mins	-97%

Table 1. Quantitative Performance Metrics

7.3 Some Threat Examples

The framework identified both common and nuanced threats across the five case studies.

7.3.1. Case A (Microservices – Fintech)

- Detected: Unauthenticated internal API (/internal/transfer) exposed to the public ingress via misconfig

ured path rewrite in the API Gateway.
- Missed (Manual Only): Incomplete rate limiting enforcement between payment retries.
- AI Insight: Identified that a database connection pool size in Terraform was insufficient for autoscaling nodes, indicating DoS risk.

7.3.2. Case B (Serverless – Media Upload)
- Detected: Public S3 bucket (acl = "public-read") storing uploaded user files without input validation.
- Detected: Lack of presigned URLs; unauthenticated users could enumerate uploaded files using predictable keys.
- Mitigation Generated: Use of presigned URLs, object-level encryption, and access logging configuration.

7.3.3. Case C (Monolith – HR Platform)
- Detected: Password reset endpoint with token expiration disabled in the config file, increasing spoofing risk.
- Missed (AI): Failure to detect long-lived JWTs being stored in localStorage.
- Strength: Accurately flagged lack of role-based access control on endpoints like /admin/employees/export.

7.3.4. Case D (Federated Cloud – IAM)
- Detected: Misuse of iam.roles.viewer granted to default service account across staging and prod.
- Detected: Hardcoded token audience value used in multiple trust relationships, increasing impersonation risk.
- AI Insight: Inferred token replay vulnerability by analyzing workload identity binding misconfigurations across namespaces.

7.3.5. Case E (ML API – Inference Engine)
- Detected: OpenAPI spec exposed full model inference interface without any rate limiting or auth guard.
- Detected: Potential prompt injection risk via user-submitted content relayed directly to an LLM backend.
- Strength: Framework suggested input sanitization and abuse detection hooks despite lack of explicit code context.

7.4. Change Responsiveness and Incremental Modeling
We simulated 90+ infrastructure and API changes (e.g., Terraform plan diffs, new endpoint PRs). For each:
- Only the affected graph segments were re-analyzed (avg. 3.5% of model updated).
- Updated threats were surfaced via structured diffs and formatted as pull request comments.
- Latency between detection and threat model update was under 120 seconds for 95% of cases.

7.5. Reviewer Feedback
Each AI-generated threat model was reviewed by two engineers per case, who rated the outputs using five criteria.

Table 2. Reviewer Feedback

Criterion	Avg. Score (1–5)	Reviewer Comments
Actionability	4.4	“Security fix suggestions were highly specific.”
Clarity	4.2	“Mitigations referenced the exact IaC block/endpoint.”
False Positive Noise	4.1	“Much lower than traditional SAST.”
Format Usability	3.9	“Easy to integrate into PR review process.”
AI Trust Level	3.8	“Some uncertainty in prioritization score logic.”

7.6. Interpretation and Summary

The case study evaluation confirms that the framework offers substantial benefits in speed, coverage, and developer alignment over traditional manual methods. Crucially, the AI was able to surface cross-layer threats—such as an OpenAPI-defined public endpoint linking to a misconfigured Terraform resource—something that is often missed in siloed SAST or IaC tools.

We observed a notable ability to adapt to different deployment types and provide developer-grade security insights, including mitigation recommendations, actionable links to code/config blocks, and inline documentation-like feedback.

However, the AI Framework showed some limitations in:

- Understanding multi-stage attack paths (e.g., auth → privilege escalation → lateral movement)
- Interpreting domain-specific semantics (e.g., how ML APIs differ in threat posture from CRUD apps)
- Providing transparent prioritization reasoning, which reviewers requested

8. Challenges and Consideration

While the proposed AI-enhanced framework offers promising scalability and alignment with modern DevSecOps workflows, several real-world challenges must be acknowledged and addressed before successful adoption. These challenges are both technical and organizational, and span the domains of data fidelity, AI accuracy, developer trust, integration complexity, and governance oversight.

8.1. Data Quality and Architecture Fragmentation

Threat modeling relies on understanding how systems are structured and how components interact. However, one of the most fundamental challenges is fragmented or outdated architectural documentation. Many engineering teams lack a centralized source of truth. Codebases, Terraform configurations, and API specs may be inconsistent, undocumented, or only partially representative of the deployed environment.

In our research, even with GitHub and Terraform as primary sources, we observed several edge cases where inferred data flows were misleading due to missing reverse proxies, internal-only endpoints not reflected in OpenAPI specs, or conditional IaC logic. Without sufficient observability into live environments, even AI-powered systems may model an inaccurate topology, leading to either missed threats or false positives.

8.2. AI Limitations and Hallucinations

While large language models have demonstrated impressive performance in tasks like summarizing code or generating security insights, they are not infallible. One critical concern is model hallucination—where the AI fabricates connections, assumptions, or vulnerabilities that don't actually exist. In high-velocity environments where decisions are made quickly, false positives stemming from hallucinated risks can erode developer trust.

Another related challenge is model interpretability. Even when the AI identifies a legitimate threat, security teams may struggle to understand how it arrived at the conclusion, especially if it involves multi-hop reasoning across code, infrastructure, and documentation. Until explainability improves, organizations must include human-in-the-loop (HITL) validation to ensure that flagged issues are contextually accurate and justified.

8.3. Incremental vs. Holistic Modeling

AI systems work best when they can process complete architectural contexts. However, modern development often involves incremental changes to specific services or modules. The challenge is twofold: (1) generating threat models that are localized to the change but still accurate in context, and (2) synchronizing that localized analysis with the global threat landscape of the entire system.

Our framework attempted to simulate such incremental updates using Terraform diff files and pull request metadata. While this proved helpful, it was not always sufficient—for instance, a new IAM policy that appeared safe in isolation could be dangerous when evaluated in conjunction with another change made a week earlier in a different repository. Addressing this requires robust state tracking and intelligent dependency graphing across services.

8.4. Developer Adoption and Contextual Relevance

Even the best threat modeling system fails if developers don't engage with it. A major consideration is the developer experience of receiving security feedback—whether it's too noisy, too generic, or too difficult to act on. Security tools historically have suffered from poor integration into developer workflows, often being relegated to PDF reports or post-release audits.

For AI-enhanced threat modeling to succeed, the delivery mechanism of insights matters as much as the detection logic. Pull request comments, GitHub code suggestions, Slack notifications, and inline IDE alerts must be contextually aware, minimal in disruption, and accompanied by clear recommendations. Developers are more likely to trust and act on suggestions that cite the exact misconfiguration, reference affected code blocks, and explain the risk in plain language.

8.5. Balancing Velocity and Rigor

One of the biggest philosophical tensions is the balance between engineering speed and security depth. Threat modeling, especially when done manually, is perceived as a blocker to deployment velocity. While our AI-augmented approach aims to make modeling lightweight and continuous, this introduces new questions around where to draw the line between quick heuristics and rigorous analysis.

For instance, in our study, low-risk infrastructure changes were auto-triaged and recommended for approval. However, without formal oversight, there's a risk of accepting changes that carry latent architectural weaknesses—like introducing a dependency loop, or making a supposedly internal API publicly accessible through future reuse. Organizational policies will need to define thresholds for AI-only decisions versus human-reviewed ones.

8.6. Toolchain Integration and Drift

Modern engineering stacks span multiple tools: GitHub, GitLab, Bitbucket; Terraform, Pulumi, AWS CDK; OpenAPI, GraphQL, and various CI/CD platforms. One major challenge is ensuring that the AI-enhanced threat modeling system is platform-agnostic yet tightly integrated enough to stay in sync with code and infrastructure changes.

Moreover, configuration drift—where the deployed environment no longer matches the IaC or code due to manual hotfixes, legacy resources, or untracked scripts—poses a serious risk. A threat model based solely on IaC and OpenAPI may miss this drift entirely unless tied into runtime validation layers or cloud inventory syncs.

9. Practical Recommendations: People, Process, and Technology

To succeed in high-velocity environments, organizations must operationalize threat modeling across three critical domains: people, process, and technology. Our research identifies phased strategies within each domain to help organizations pragmatically scale their security posture without compromising deployment velocity.

9.1. People

In the short term, organizations should consider expanding dedicated product security teams and actively investing in the growth of Security Champions within engineering groups [4]. These champions bridge the gap between security and development and serve as early adopters of tooling and processes. Medium-term, security teams should run recurring open office hours to coach developers and champions, and offer targeted enablement around threat modeling tooling—such as threat mapping workflows, STRIDE modeling, or IaC diff review. Training these champions on AI-augmented platforms also allows them to contribute threat analysis with less dependency on centralized security resources. In the long term, large language models can automate triaging of low-risk design changes or suggest first-pass mitigations for common patterns, allowing human experts to focus on complex, novel risks.

9.2. Process

Effective process design begins with simplifying the threat modeling intake and embedding it in existing workflows (e.g., pull request templates, sprint planning). Teams should define clear triggers and acceptance criteria for when a threat model is required—for example, a new external-facing API or a change to authentication logic. Medium-term efforts should focus on aligning intake workflows across teams and creating predictable feedback loops by assigning SLAs for threat model turnarounds. Risk triaging based on automation (e.g., tagging IaC modules as low, medium, or high-risk) helps focus effort where it matters. Long-term, organizations should develop reference threat models and secure-by-default architectural patterns that engineers can adopt and extend. These templates accelerate development and reduce security review overhead, while still encouraging system-specific analysis where deviations occur.

9.3. Technology

In the short term, organizations should focus on delivering contextual, just-in-time feedback via a centralized messaging layer—integrated with Slack, GitHub pull requests, or dashboards. This allows security findings to be immediately actionable without derailing developer momentum. Research also highlights the need for onboarding portals that define the scope, SLAs, and support model for threat modeling. In the medium term, metrics collection across code changes, threat annotations, and incident correlations can drive investment decisions and tool refinement. Integrations with CI pipelines and secrets scanning tools can provide early signal on insecure changes. In the long term, incorporating runtime protection mechanisms such as Runtime Application Self-Protection (RASP) and Interactive Application Security Testing (IAST) [9] can bridge gaps between design-time assumptions and real-world application behavior. Finally, developer-centric security tooling such as IDE plugins and AI assistants can further scale security context by surfacing risks early during code authoring.

9.4. Phased Rollout Strategy

To ensure sustainable adoption, the rollout of AI-enhanced threat modeling should follow a carefully staged approach that aligns with organizational readiness and maturity.

9.4.1. Phase One (Months 1–6)

Phase 1 centers on piloting the approach with a small number of high-impact services or teams. During this phase, security teams should collaborate with engineering leads to select representative use cases—such as the onboarding of a new public API, or refactoring of infrastructure using Terraform modules. A network of Security Champions should be formally established to act as early adopters and feedback conduits. These champions will work closely with the AI-enabled tooling and report usability insights, accuracy gaps, and operational friction. GitHub repositories, Terraform configurations, and OpenAPI specs are ingested to simulate threat models in a controlled environment, with human-in-the-loop (HITL) validation. The focus is not on complete automation but on demonstrating early value and learning from real workflows. Documentation, FAQs, and training modules are also developed during this period.

9.4.2. Phase Two (Months 7–10)

This phase is dedicated to scaling adoption across more teams and services while strengthening integration depth. AI-assisted threat modeling is embedded directly into CI/CD workflows using GitHub Actions, Slack bots, and merge request templates. Intake processes are formalized with clear acceptance criteria, threat model triaging logic, and response SLAs. Feedback loops are codified via structured surveys, post-mortems, and pull request metrics. The AI models are also tuned using feedback gathered during Phase One, reducing hallucinations and improving contextual sensitivity. At this stage, governance functions—such as audit trails for accepted/ignored risks and policy deviations—are integrated into existing risk registers or GRC tooling. Metrics such as percentage of code changes covered by a threat model, accuracy of risk categorization, and time to threat model generation are tracked consistently.

9.4.3 Phase Three (Months 10–15)

Final Phase focuses on optimization, reliability, and continuous learning. Threat modeling systems evolve toward self-healing behaviors, where AI detects drift between architecture and code and updates models automatically. Explainable AI layers are introduced to provide rationale for prioritization or mitigation suggestions. Insights are federated across services—such that learnings from one domain (e.g., authentication flows) inform others (e.g., session management in microservices). Security steering committees composed of engineering, product, and security leadership are formed to ensure alignment, set quarterly goals, and respond to systemic gaps. Integration with runtime signals (e.g., from Runtime Detection or anomaly detection tools) enables real-world feedback to refine model assumptions.

Key performance indicators (KPIs) tracked throughout the rollout include:

- Threat model coverage rate across code and infrastructure changes
- Time reduction from threat model request to insight delivery
- Accuracy of AI-generated threat detection compared to manual reviews
- Developer satisfaction with AI feedback (via post-PR surveys)
- Mean time to remediate (MTTR) threats detected during development

By monitoring these metrics, organizations can evaluate the return on investment (ROI) of the AI-enhanced approach and make data-informed decisions on where to refine or expand capabilities. A successful rollout does not depend on instant perfection, but on measured iteration, cross-functional buy-in, and clear feedback loops across engineering and security stakeholders, all of which build trust and continuity in the security development lifecycle.

Ultimately, by aligning initiatives across these three dimensions—people, process, and technology—and executing the phased rollout strategy, organizations can create a scalable, developer-aligned threat modeling program. This program not only improves coverage and accuracy but also embeds security into the fabric of rapid delivery cycles, ensuring threat modeling remains a value driver rather than a velocity bottleneck.

10. Conclusions

As development velocity increases under DevSecOps practices, traditional threat modeling approaches—manual, static, and often disconnected from engineering workflows—are no longer sustainable. This paper presented an AI-enhanced framework that reimagines threat modeling as a continuous, context-aware process integrated directly into CI/CD pipelines. By leveraging large language models to extract architectural understanding from source artifacts like GitHub, Terraform, and OpenAPI, the framework automates threat identification and prioritization at scale.

Rather than treating threat modeling as a point-in-time review, our framework synchronizes with real-time system changes, ensuring security analysis evolves alongside software delivery. Evaluation across five production-inspired architectures showed measurable gains: threat modeling time was reduced by 83%, endpoint coverage improved by 42%, and detection of high-risk design flaws increased by 26% in post-engagement red teaming. These results demonstrate the practical value of augmenting security workflows with AI to increase precision, coverage, and responsiveness without overburdening engineering teams.

Successful adoption, however, is not purely technical. Embedding threat modeling into developer workflows, enabling feedback in real time, and empowering Security Champions to act as local reviewers are essential to driving sustained cultural change. AI serves here not as a replacement for human expertise, but as a force multiplier—automating rote analysis, surfacing architectural drift, and freeing up security teams to focus on strategic risk.

In closing, we argue that AI-assisted threat modeling is not just a toolset but a shift in how security is operationalized. With the right integrations, processes, and oversight, this approach can close the critical gap between secure design and fast-paced development—and set the foundation for security to scale with engineering.

References

1. OWASP Foundation. (2023). *OWASP Threat Modeling*. Available: https://owasp.org/www-project-threat-modeling/
2. Shostack, A. (2014). *Threat Modeling: Designing for Security*. Wiley Publishing, Inc.
3. Fitzgerald, G., & Price, C. (2020). “DevSecOps and Automated Security Testing,” *IEEE Software*, vol. 37, no. 4, pp. 14–18, doi: 10.1109/MS.2020.2981225.
4. Fortuna, A. (2020). “DevSecOps: the value of Security Champions,” *DevSecOps Blog*. Available: https://andreafortuna.org/2020/01/23/devsecops-the-value-of-security-champions/

5. Microsoft Corporation. (2022). “Security Development Lifecycle (SDL) Practices.” Available: https://learn.microsoft.com/en-us/security/sdl
6. Tomas, D., et al. (2021). “Mining GitHub Repositories for Predictive Security Insights,” in *Proc. ACM MSR*, pp. 134–145.
7. Shah, A., & Bansal, R. (2022). “Threat Modeling for Infrastructure-as-Code,” in *Proc. OWASP Global AppSec*, San Francisco.
8. Postman, Inc. (2023). “Security Risks in OpenAPI Specifications,” *Postman Security Research*. Available: https://www.postman.com
9. Lal, S., & Tripathi, A. (2022). “Modern Application Protection using IAST and RASP,” *IEEE Cybersecurity Development Conference*.
10. MITRE Corporation. (2023). *CVE Program – Vulnerability Data*. Available: https://cve.mitre.org
11. FIRST. (2021). *Exploit Prediction Scoring System (EPSS)*. Available: https://www.first.org/epss/
12. Chowdhury, A., & Pham, T. (2023). “LLM Guardrails in Secure Software Engineering,” in *Proc. ACM CCS Workshop on AI Safety*.
13. Sion, L., & Carter, L. (2022). “The Role of AI in Security Automation: From Reactive to Proactive,” *IEEE Security & Privacy*, vol. 20, no. 3, pp. 72–80, doi: 10.1109/MSEC.2022.3147986.
14. Haney, J., & Lueck, G. (2020). “Measuring the Effectiveness of Threat Modeling in Agile Development Environments,” *Journal of Cybersecurity*, vol. 6, no. 1, doi: 10.1093/cybsec/tyaa005.
15. Scandariato, R., Walden, J., & Joosen, W. (2015). “Static Analysis Versus Penetration Testing: A Controlled Experiment,” *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 1, doi: 10.1145/2809789.
16. Zeller, A., & Wüchner, T. (2023). “From Secure by Design to Secure at Runtime: The Role of ML-based Threat Analysis,” *ACM CCS Workshop on AI and Security*.
17. Sabottke, C., Suciu, O., & Dumitras, T. (2015). “Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits,” *Proc. USENIX Security Symposium*.
18. Rakitin, S. (2017). “Case Study: Implementing Threat Modeling Across a Financial Institution,” *SANS Institute Whitepaper*.
19. ENISA. (2021). *Artificial Intelligence Threat Landscape*. European Union Agency for Cybersecurity. Available: https://www.enisa.europa.eu/publications/artificial-intelligence-threat-landscape
20. Xiong, Y., & Yu, W. (2023). “Secure Software Development with AI: Challenges and Opportunities,” *IEEE Trans. Softw. Eng.*, early access, doi: 10.1109/TSE.2023.3254906.
21. Shostack, A. (2023). “Next-Generation Threat Modeling: Where Do We Go from Here?” *Black Hat USA*. Available: https://www.blackhat.com/us-23/briefings/schedule/#next-generation-threat-modeling-where-do-we-go-from-here-30593