GraphIDE

AUTONOMOUS, VERIFIABLE AI SECURITY FOR MODERN SOFTWARE

Team - Trust1ssues

Closing the AI “Vibe Coding” Credibility Gap

Trust1ssues

Problem Statement

Background and Overview

Context

AI-assisted development enables rapid generation and modification of large codebases. Security verification has not scaled at the same pace, leaving AI-written logic insufficiently understood before deployment.

Core Problem

Most security tools rely on pattern matching or probabilistic text inference. They cannot deterministically prove whether a vulnerability is reachable in real execution paths, leading to unreliable results.

Impact

Unverified vulnerabilities may reach production while security teams are overwhelmed by noise. Organizations lack repeatable, auditable proof of security validation, increasing operational and compliance risk.

Key Analytics & Industry Statistics

- 65%+ developers **reuse** AI-generated code with minimal or no review
- AppSec teams spend 40–60% of their time **triaging false positives**
- Average cost of a **security breach**: $4.5M+ per incident
- Code volume is growing 3–5× faster than security review capacity
- Enterprise codebases now contain **millions** of AI-modified lines monthly

Who Is Affected?

- **Developers**: Unsure whether AI fixes are safe or will break builds
- **AppSec Engineers**: Overwhelmed by alerts, forced to manually validate findings
- **CTOs & Security Leaders**: Unable to confidently approve AI-generated code for production
- **Compliance Teams**: No permanent audit trail or proof of due diligence

“Modern software is written by AI, but secured by tools that were never designed to verify AI-generated logic.
This is the gap Graphide is built to solve.”

Trust1ssues

GraphIDE

Graphide closes the gap by transforming AI security from a conversational assistant into a fully automated, deterministic verification and remediation pipeline.

Solution Overview

- Graph-Based Security Verification - Analyzes code using structural program graphs to reason about real control-flow and data-flow behavior.
- Progressive Code Reduction - Incrementally narrows large codebases into small, security-relevant execution paths for focused analysis.
- Proof-Guided AI Reasoning - Applies AI models only after vulnerabilities are structurally validated, preventing speculative or hallucinated findings.
- Automated Vulnerability Remediation - Classifies issues using CWE standards and generates validated fixes that integrate safely into the codebase.
- Audit-Ready Security Outputs - Produces visual evidence, structured explanations, and persistent reports for enterprise and compliance use.

Unique Selling Proposition

- **Proof-before-reasoning** architecture ensures vulnerabilities are structurally verified before any AI reasoning is applied.
- **Code slicing funnel** progressively **reduces the codebase**, minimizing false positives, analysis noise, and infrastructure cost.
- **AST-validated**, **auto-applied patches** guarantee syntactically correct fixes that integrate safely without breaking builds.

Why does it work?

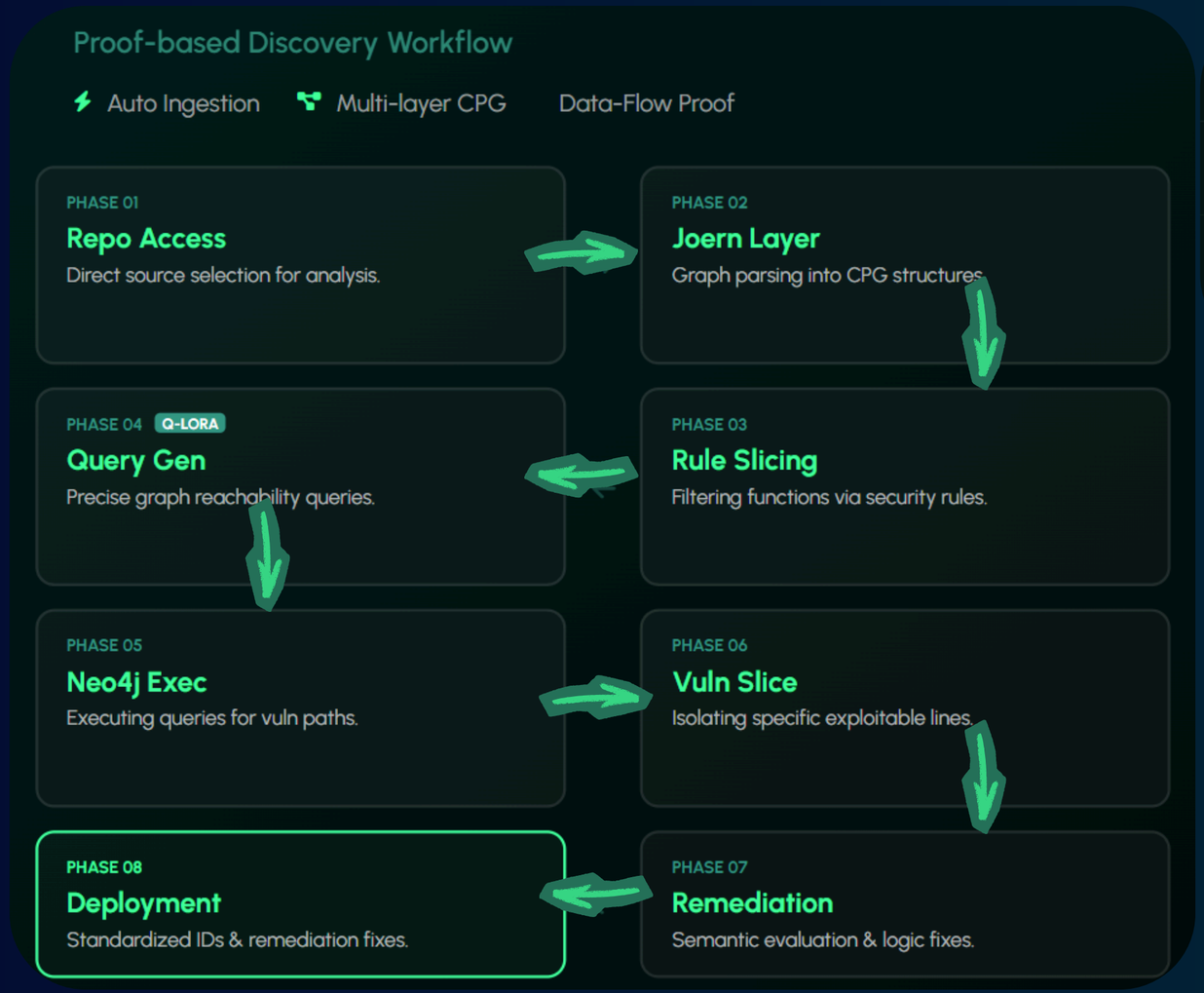
Proof before AI reasoning ensures vulnerabilities are real, not inferred.

Progressive code slicing removes noise and scales analysis efficiently.

Bounded AI reasoning produces accurate, reliable fixes on minimal code.

Trust1ssues

Technical Approach



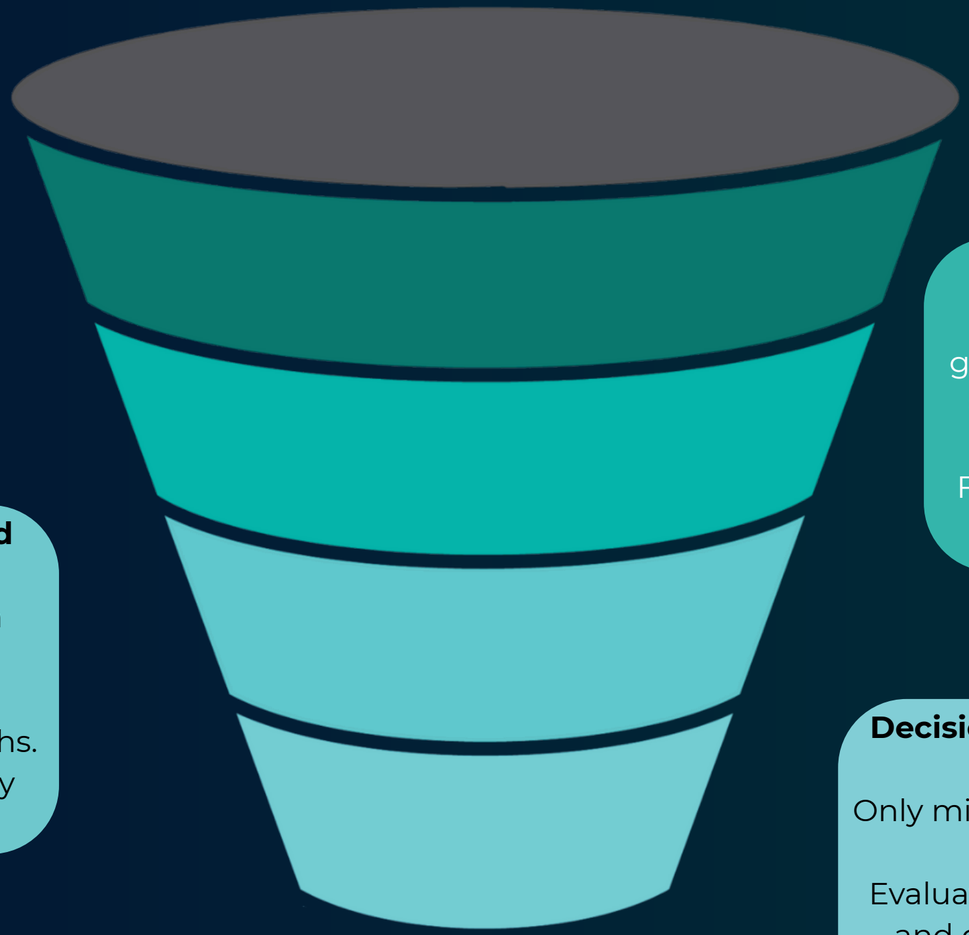
CRITERIA	CLAUDE	GPT	GRAPHIDE
Tokens Used	Claude doesn't take this data	~3,800,000	27,000
Accuracy	—	≈ 20–25%	81% (FormAI) 72% (PrimeVul)

CWE	Accuracy
CWE-119	0.941
CWE-415	0.757
CWE-416	0.778
CWE-190	0.672

Trust1ssues

Funnel-based Approach

Static Layer – Rule-Based Slicing
Performs a full repository scan using predefined security rules.
Identifies and extracts potentially vulnerable functions and code regions.
Eliminates clearly safe code early to reduce analysis scope.



CPG Query Generating Model
An LLM fine-tuned with **Q-LoRA** generates targeted Joern CPG queries.
Queries are derived from sliced code context and vulnerability patterns.
Focuses on data-flow and control-flow reachability validation.

Graph-Based Slicing via Generated Queries -
Generated queries are executed on Joern-built Code Property Graphs.
Further slices code by proving or eliminating vulnerable execution paths.
Reduces analysis to only structurally reachable paths.

Decision / Reasoning Model (Gemini API)
Only minimal, high-risk lines of code are forwarded for reasoning.
Evaluates exploitability, classifies CWE, and determines remediation logic.
Prevents speculative or hallucinated vulnerability analysis.

Trust1ssues

Proof-based Discovery Workflow

Auto Ingestion Multi-layer CPG Data-Flow Proof

PHASE 01
Repo Access
Direct source selection for analysis.

PHASE 02
Joern Layer
Graph parsing into CPG structures.

PHASE 03
Rule Slicing
Filtering functions via security rules.

PHASE 04 Q-LORA
Query Gen
Precise graph reachability queries.

PHASE 05
Neo4j Exec
Executing queries for vuln paths.

PHASE 06
Vuln Slice
Isolating specific exploitable lines.

PHASE 07
Remediation
Semantic evaluation & logic fixes.

PHASE 08
Deployment
Standardized IDs & remediation fixes.

Trust1ssues

Technical Workflow

- **Graph-Centric Program Representation** - Source code is transformed into Code Property Graphs (AST + CFG + PDG) to enable precise structural and semantic analysis.
- **Rule-Based Code Slicing** - Static security rules extract potentially vulnerable functions, reducing analysis scope early and deterministically.
- **LLM-Assisted Graph Query Generation** - A fine-tuned Q-LoRA model generates targeted Joern CPG queries for reachability and data-flow **validation.**
- **Deterministic Graph Execution** - Generated queries are executed on Joern with Neo4j-backed graphs to prove vulnerability reachability.
- **Proof-Guided AI Reasoning** - Only minimal, verified code slices are forwarded to reasoning models for CWE classification and remediation.
- **AST-Validated Automated Fixes** - Generated patches are validated against the program's AST and safely applied to the codebase.

Token & Cost Efficiency - Traditional LLM-based security tools repeatedly analyze entire codebases, consuming large token volumes.
Graphide avoids this by:

- Applying AI only on **minimal vulnerable slices**
- Eliminating **speculative reasoning**
- **Stopping analysis early** when no path exists

Trust1ssues

Why existing solutions don’t work?

Core Issues with GPT/Claude & Traditional Tools

- LLM-only tools analyze code as text and infer risk patterns, leading to hallucinations and unreliable findings.
- Pattern-based static analysis cannot prove data-flow reachability or true exploitability.
- They produce high false positive rates, slowing down security teams and eroding trust.
- They lack audit-ready, deterministic evidence needed for enterprise compliance.

Real Metrics: Industry Performance Flow

1. False Positives

- LLM-ONLY
  ~25–40%
- TRADITIONAL SAST
  ~30–50%
- GRAPH-BASED VALIDATION
  ~<5% (Verified)

2. Patch Accuracy

- LLM-ONLY PATCHES
  Compile ~60–75%
- AST-VALIDATED AUTO PATCHES
  ~99%+ (Graphide)

3. Actionable Result

- TRADITIONAL TOOLS
  2–4 hrs + manual review
- LLM AGENTS
  45–90 min (unverified)
- GRAPHIDE
  8–15 min End-to-End

Comparison Table: Existing Tools vs Graphide

CRITERIA	GPT/CLAUDE (LLM-ONLY)	TRADITIONAL TOOLS (SAST/AST)	GRAPHIDE
Reasoning Basis	Text inference	Pattern matching	**Graph reachability proof**
Hallucination Risk	**High**	Medium	**Minimal**
False Positives	~25–40%*	~30–50%*	**<5%**
Patch Reliability	Often broken	N/A / Manual	**AST-validated, auto-applied**
Execution Path Proof	✗	✗	✓✓
Visual Evidence	None	Limited	**Interactive graphs**
Compliance Reporting	None	Limited	**Audit-ready PDFs**
Cost Efficiency	High token use	Low logic depth	**Efficient funnel-based**

Critical Bottlenecks In Existing Tooling

LLM Tools

- No reliability in data-flow or exploitability proof
- Hallucinate vulnerabilities and unstable patches
- Excessive token costs on context re-processing

Traditional Static

- Pattern scanning lacks true semantic reasoning
- High noise environments and low precision
- Limited support for complex dynamic paths

Combined / Hybrid

- Inefficient repeated analysis of entire codebases
- Premature application of AI logic/reasoning
- Wasted compute without structural proof first

Trust1ssues

Use-Case and Impact

Primary Use Cases

AI-Assisted Development Security - Validate AI-generated code (Copilot, ChatGPT, internal LLMs) before merge, ensuring only provably safe logic enters production.

Enterprise Application Security (AppSec) - Replace noisy static analysis with graph-verified findings, allowing security teams to focus on real, exploitable vulnerabilities

Large Codebase Audits & Legacy Modernization - Ingest entire repositories to uncover hidden data-flow vulnerabilities in legacy or poorly documented systems

Open-Source Risk Assessment - Scan external dependencies or GitHub projects prior to adoption, upgrades, or M&A technical due diligence.

Enterprise Pipeline Integration

Graphide is designed to plug directly into existing engineering workflows:

IDE / AI Coding Tool
→ Git Push / PR
→ CI Pipeline (GitHub Actions, GitLab CI, Jenkins)
→ Graphide Security Scan
→ Verified Findings + Auto-Fix
→ Merge Gate / Deployment

Business & Security Impact

- Reduced Security Risk - Prevents exploitable vulnerabilities from reaching production by validating real execution paths.
- Lower Operational Cost - Cuts false positives and manual triage time by up to 70–90%, improving AppSec efficiency.
- Faster Time to Release - Automated, verified fixes reduce security bottlenecks without slowing development.
- Compliance & Audit Readiness - Produces persistent, traceable security artifacts required for SOC2, ISO, and internal audits.

Trust1ssues

Feasibility and Viability

Technical Feasibility

- Built on proven technologies such as static analysis, Code Property Graphs (Joern), and graph databases (Neo4j).
- LLM usage is bounded and targeted, making the system stable, predictable, and scalable.
- Modular architecture allows incremental rollout (scan-only → auto-fix → reporting).
- Successfully handles large repositories through progressive code slicing.

Operational Feasibility

- Fully automated pipeline requires minimal human intervention.
- Can run as a CI/CD stage, standalone service, or internal security tool.
- Integrates cleanly with existing developer workflows and repositories.
- Designed to work alongside, not replace, existing security tooling.

Economic Viability

- Funnel-based design results in 5–10× lower LLM token usage compared to agent-based tools.
- Reduced false positives significantly lower AppSec triage and remediation costs.
- Predictable infrastructure costs make it enterprise-friendly at scale.

Business Viability

- Addresses a growing need as AI-generated code becomes standard across enterprises.
- Strong fit for mid-to-large organizations with security, compliance, and AI adoption requirements.
- Applicable across industries including finance, healthcare, SaaS, and regulated sectors.

Long-Term Viability

- Architecture is language-agnostic and extensible.
- Can expand into multi-language support, CI-native security gates, and compliance automation.
- Positioned as a foundational security layer for AI-driven software development.

Trust1ssues

Future Scope

Advanced Capabilities & Extensions

1. Patch Application Agent
Graphide includes an automated patch application agent that applies verified security fixes directly to the codebase.
Each patch is validated against the program's Abstract Syntax Tree (AST) to ensure syntactic correctness and build safety.
This enables near-instant remediation without manual developer intervention.

2. Graph-Based Vulnerability Visualization
Every verified vulnerability is accompanied by an interactive graph visualization showing Source → Sink execution paths.
Built on Neo4j, these graphs provide visual proof of exploitability, enabling faster validation and debugging.
This improves trust, explainability, and auditability for security teams.

3. Automated Dynamic Analysis & Red Teaming
Graphide extends static verification with automated red-team-style attacks executed in a sandboxed environment.
The system simulates real-world cyberattacks to validate exploitability under runtime conditions.
This bridges the gap between static analysis and real execution behavior.

4. End-to-End Vulnerability Analysis & Reporting
Graphide delivers a fully automated workflow from code ingestion to remediation.
Each scan produces structured security reports including vulnerabilities, graphs, patches, and risk classification.
Reports are audit-ready and suitable for enterprise compliance and governance.

5. Multi-Language & Polyglot Support
Graphide is designed to support multiple programming languages, including **Go, Rust, JavaScript, and Python**, through language-specific frontends and CPG generation.
This enables consistent vulnerability analysis, verification, and remediation across modern polyglot codebases.
The architecture is extensible, allowing new language support to be added without changes to the core verification pipeline.

Thank You