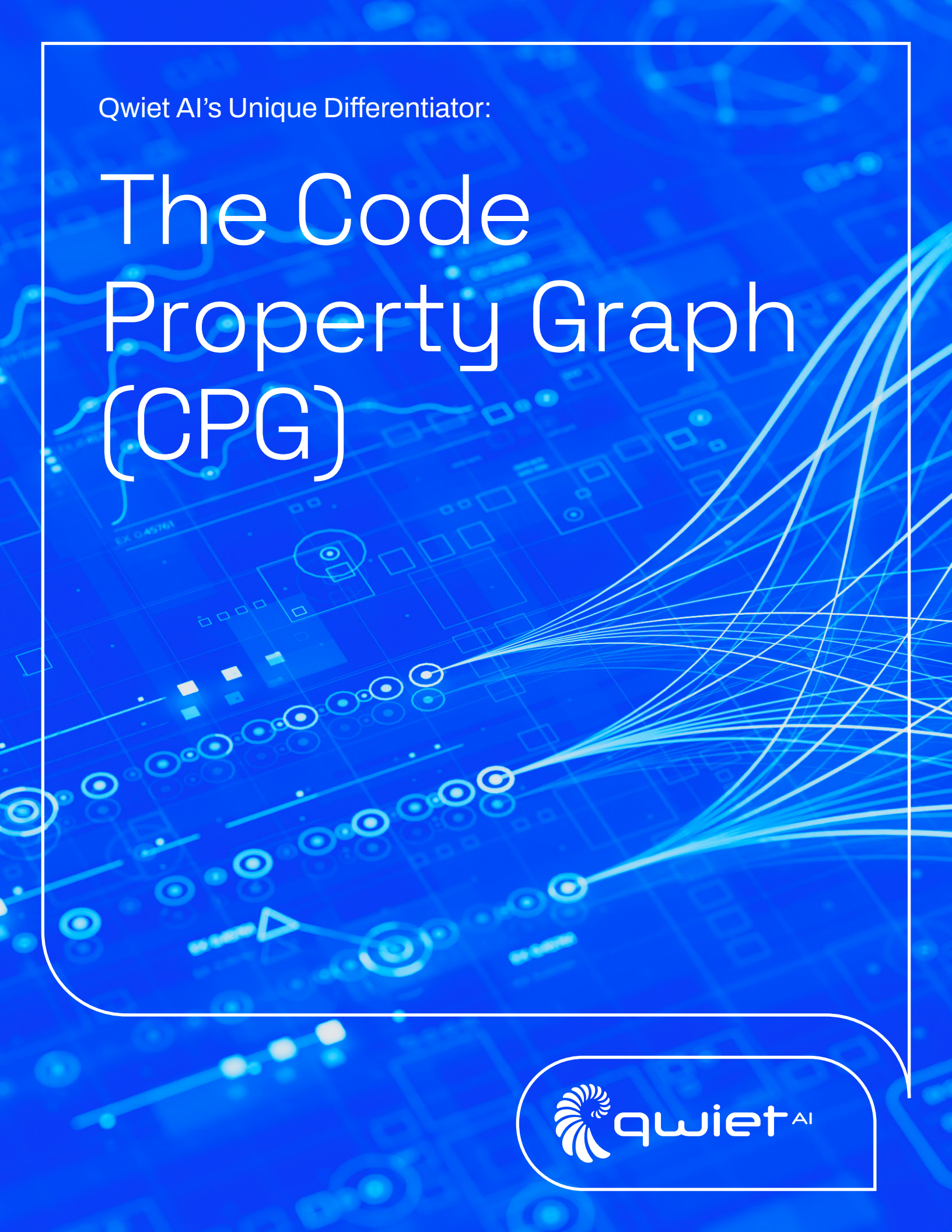
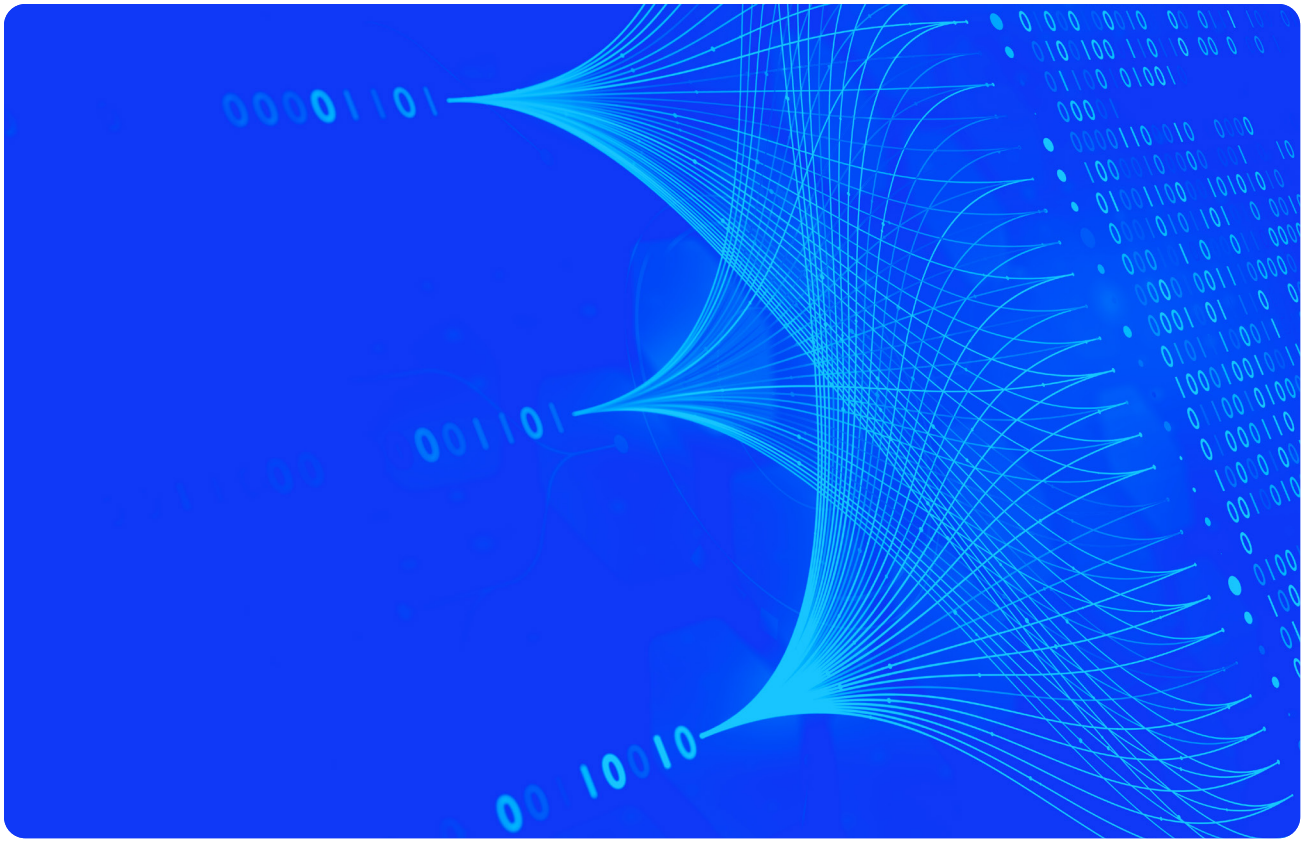
Qwiet AI's Unique Differentiator:

The Code Property Graph (CPG)



qwiet AI



Introduction

Artificial intelligence and machine learning are revolutionizing numerous fields, including source code. Developers are experiencing the transformative capabilities of tools like GitHub Copilot, which move beyond simple autocomplete to offer intelligent code generation. However, these tools typically operate on **isolated snippets**, limiting their impact on productivity, quality, and security.

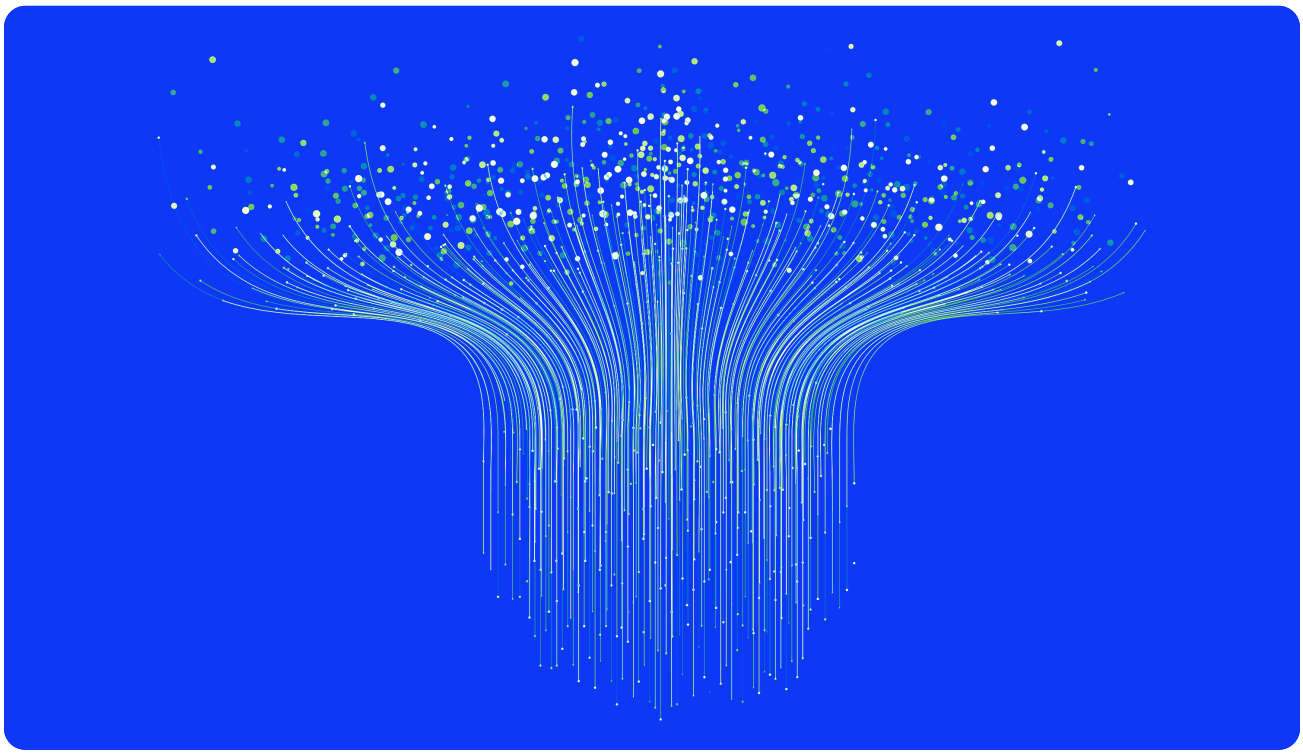
Qwiet's **Code Property Graph (CPG)** is a game-changer that overcomes this critical limitation. CPG provides a **semantically rich, graph-based representation of entire codebases**, empowering AI to reason not just about lines of code but system-wide behavior. This revolutionary capability positions CPG-enhanced AI as a foundational advancement in software engineering, unlocking deeper insights and automation that today's token-based models can't match.Agentic AI: Reinforcing Security Guardrails

The Current Problem

Despite the advances in AI coding tools, development teams still face significant limitations:

- **Blind code generation** that ignores architectural context
- **Security vulnerabilities** that go undetected due to a lack of data flow awareness
- **Fragmented optimization**, where AI tools can't see beyond the file-level scope
- **Repeated context sharing**, where developers constantly re-explain intent

These challenges stem from how AI sees code. Traditional tools treat source code **as plain text or sequences of tokens**—a method that, while useful, lacks awareness of syntax structure, semantics, and interrelationships between code components.



The business impact is real:

- Up to **40% of AI-generated code** requires rework
- Security issues **routinely escape detection**.
- Development and onboarding timelines are **longer than necessary**.

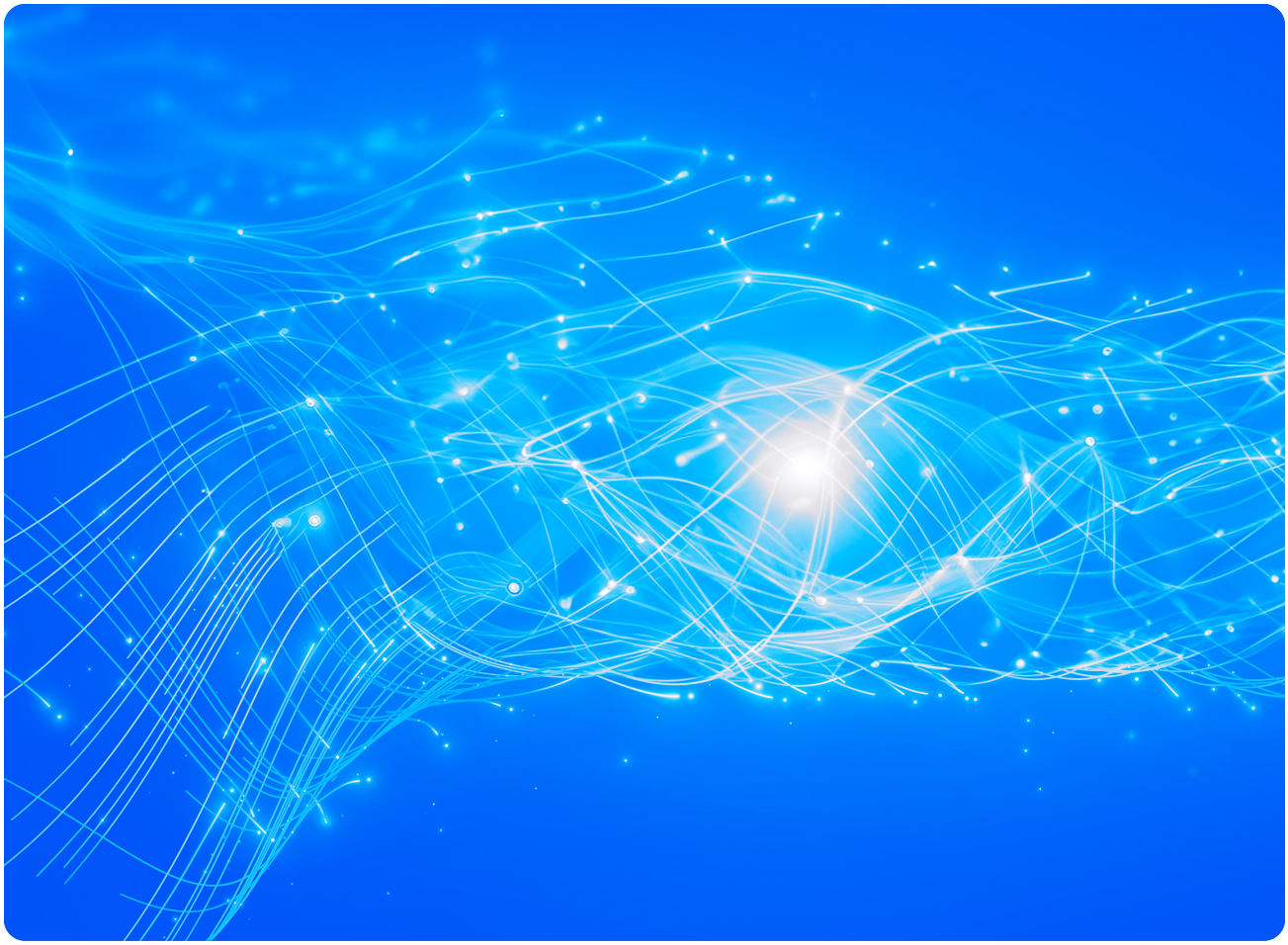
Why Code Needs Graphs

Unlike natural language, **source code has formal structure, data dependencies, and control flows**. Capturing this requires more than tokenization or normalization; it demands a structured representation that reflects the real logic of code.

Graph-based representations provide this structure:

- **Abstract Syntax Trees (ASTs)** reflect syntactic structure
- **Control Flow Graphs (CFGs)** model execution paths
- **Program Dependence Graphs (PDGs)** capture data and control dependencies

Qwiet's distinct advantage lies in its **Code Property Graph**. Unlike approaches that analyze programs through separate, single-faceted graphs, Qwiet's **Code Property Graph** provides a **multi-dimensional understanding** of system behavior. It achieves this by unifying the **Abstract Syntax Tree (AST), Control Flow Graph (CFG), and Program Dependence Graph (PDG)** into a single, searchable structure.



The CPG Advantage

Qwiet's CPG creates a **compressed semantic map** of the codebase, enabling AI to reason holistically. This results in:

- **Immediate ROI:** Reduce rework cycles by over **60%** through context-aware code generation that aligns with architectural intent
- **Risk Reduction:** Prevent vulnerabilities by tracing **untrusted data flows** and flagging unsafe code patterns before they ship.
- **Faster Time-to-Market:** Enable confident large-scale refactoring and modernization
- **Cost Efficiency:** Identify cross-cutting optimization opportunities and reduce technical debt.

This representation also serves as a **natural input for Graph Neural Networks (GNNs)**, making it future-proof for ML-driven source code classification, vulnerability detection, and behavior modeling.

Business Impact Areas

Developer Productivity

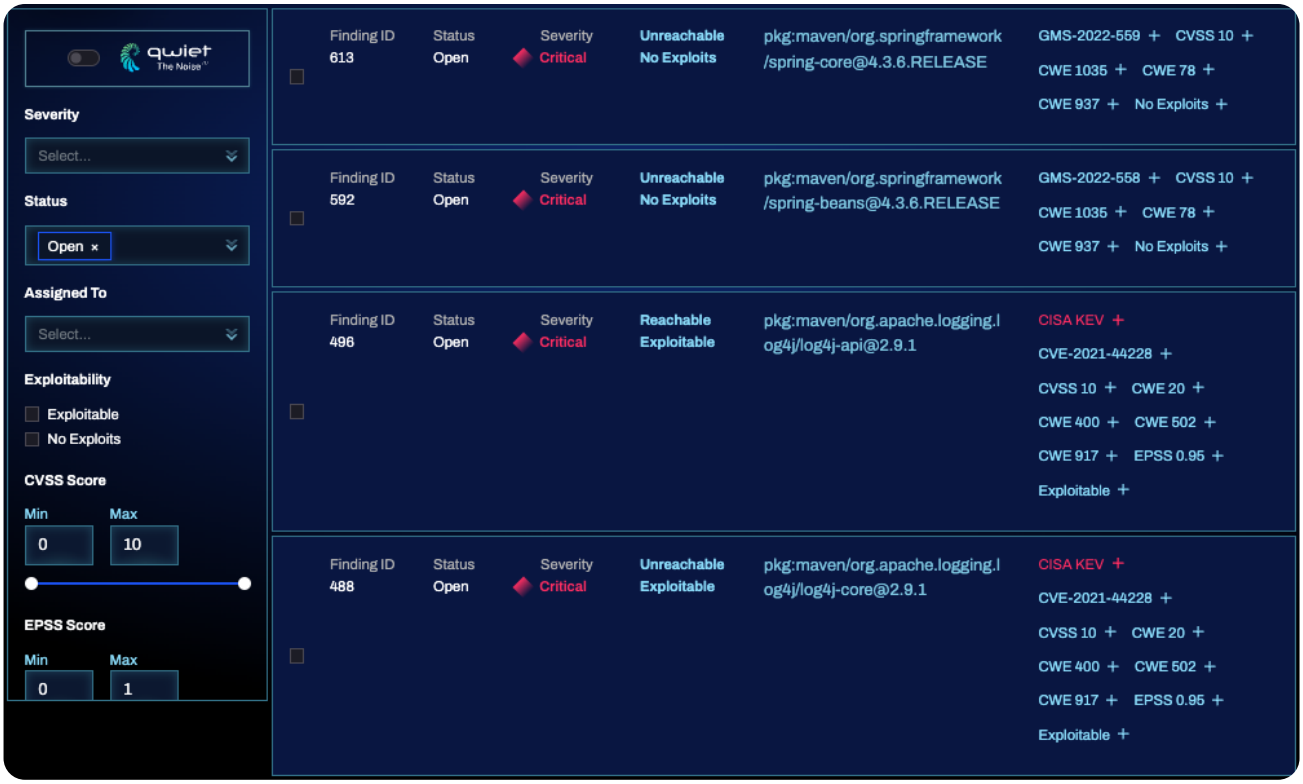
Where traditional AI tools offer modest gains, CPG-driven tooling delivers:

- **30-40% faster onboarding** by allowing new developers to explore the whole system via intelligent graph navigation
- **25% time savings** on repetitive coding tasks thanks to better context
- **50% speedier comprehension** of complex architectures through graph-driven exploration

Security Posture

Post-release vulnerability fixes cost **6x more** than preemptive mitigation. CPG empowers secure development by:

- **Proactively identifying vulnerabilities** through semantic data flow analysis
- **Reducing security-related bugs by 35%** with pattern-based prevention
- **Improving audit outcomes** by enforcing security policy adherence in generated code



Competitive Edge

Organizations adopting CPG-enabled development benefit from:

- **Faster responsiveness** to market demands
- **More reliable releases** with fewer critical issues
- **Aligned feature development** with architectural constraints
- **Improved developer retention** through the elimination of frustrating, low-context AI rework

Implementation Roadmap

- **Quick Wins:** Integrate CPG with existing LLM-based tools in targeted workflows
- **Scale:** Apply CPG to security-critical systems where ROI is highest
- **Standardize:** Make CPG-enhanced development tooling the norm across the engineering organization.

Bottom Line

The difference between today's AI code assistants and a CPG-enhanced system is the difference between a typist and an architect. **Code Property Graphs enable systems that understand, not just generate, code.**

CPG isn't optional for organizations serious about leveraging AI in software development; it's **the foundation** for realizing AI's **actual business value**: faster delivery, more secure code, and dramatically improved engineering effectiveness.

See the CPG in Action

Schedule a 30-minute demo to learn how CPG-enhanced AI identifies vulnerabilities, reduces rework, and accelerates your development lifecycle.

References

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). *A survey of machine learning for big code and naturalness.* ACM Computing Surveys (CSUR), 51(4), 1–37. https://doi.org/10.1145/3212695

Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. (2014, May). *Modeling and discovering vulnerabilities with code property graphs.* In 2014 IEEE Symposium on Security and Privacy (pp. 590–604). IEEE. https://doi.org/10.1109/SP.2014.44

Hellendoorn, V. J., Tu, Z., Sutton, C., & Raychev, V. (2021). *Global context in neural code completion.* Proceedings of the ACM on Programming Languages, 5(OOPSLA), 1–29. https://arxiv.org/abs/2005.02530

National Institute of Standards and Technology (NIST). (2002). *The economic impacts of inadequate infrastructure for software testing.* U.S. Department of Commerce. https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf

Veracode. (2023). *State of software security report.* https://www.veracode.com/state-of-software-security-report

GitHub. (2023, March 1). *The impact of Copilot on developer productivity.* GitHub Blog. https://github.blog/2023-03-01-research-experiment-copilot-productivity-statistically-significant/

McKinsey & Company. (2022). *The state of AI in 2022 and a half-decade in review.* https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2022-and-a-half-decade-in-review

Leskovec, J. (2024). *CS224W: Machine learning with graphs.* Stanford University. https://web.stanford.edu/class/cs224w/

Sajnani, H., Saini, V., Svajlenko, J., Roy, C. K., & Lopes, C. V. (2016, May). *SourcererCC: Scaling code clone detection to big code.* In Proceedings of the 38th International Conference on Software Engineering (ICSE) (pp. 1157–1168). IEEE. https://doi.org/10.1145/2884781.2884877