Morph Management

SECURITY RISKS USING LLMs IN SOFTWARE DEVELOPMENT

Risks, Controls, and Best Practices for Enterprise AI-Assisted Development

March 2026

Table of Contents

2 EXECUTIVE SUMMARY 3

3 THE REAL AND EMERGING RISKS 4

3.1 Secrets and Credentials Leaking into Training Data 4
3.2 Code Sent to LLM Providers Becomes an Exposure Surface 5
3.3 Vulnerable Code Generation 5
3.4 Supply Chain Poisoning via Hallucinated Dependencies 5
3.5 Agent and Tool Vulnerabilities 5
3.6 Membership Inference and Model Inversion 6

4 THE LITELLM + PANGEA APPROACH 6

4.1 LiteLLM: The Gateway Layer 6
4.2 Pangea AI Guard: The Content Scanning Layer 7
4.3 How They Work Together 8
4.4 Alternative and Complementary Solutions 9
4.4.1 Alternative LLM Gateways 9
4.4.2 Alternative Content Scanning and Data Lost Prevention (DLP) Solutions 9
4.4.3 Complementary Controls 10

5 BEST PRACTICE: CONTROLS, NOT VOLUNTARY BEHAVIOUR 11

5.1 Network and Gateway Layer (Enforced) 11
5.2 Development Environment Layer (Enforced) 11
5.3 CI/CD Pipeline Layer (Enforced) 12
5.4 Governance and Audit Layer (Enforced) 12

6 STATISTICS, INDUSTRY ASSESSMENT, & OUTLOOK 13

6.1 Key Statistics (2024–2026) 13
6.2 Regulatory and Standards Landscape 13
6.3 How the Threat Landscape Is Evolving 14
6.3.1 Near-term (2026–2027) 14
6.3.2 Medium-term (2027–2029) 14
6.3.3 Long-term considerations 14

7 SUMMARY RECOMMENDATIONS 15

8 APPENDIX A: REFERENCES 16

8.1 Section 1: The Real and Emerging Risks 16
8.2 Section 2: LiteLLM + Pangea AI Guard 17
8.3 Section 3: Alternative and Complementary Solutions 17
8.4 Section 4: Best Practices and Controls 18
8.5 Section 5: Statistics and Industry Assessment 19

1 Executive Summary

The rapid adoption of AI-assisted software development – with up to 90% of developers using LLM-powered coding tools by 2025 – has created a significant and growing security risk that most organisations are not adequately controlling. This document examines the principal threats, evaluates the LiteLLM and Pangea AI Guard architectural approach, and sets out an enforceable controls framework.

The core risks are interconnected. LLMs trained on public web data have absorbed millions of examples of hardcoded secrets, causing them to reproduce insecure patterns by default. Code and configuration sent to cloud-hosted AI providers expands an organisation's exposure surface, even where providers contractually limit training use. Academic and industry research consistently shows that 43–45% of AI-generated code contains security flaws drawn from the OWASP Top 10, and a growing attack class – "slopsquatting" – exploits the tendency of LLMs to hallucinate package names by pre-loading those phantom packages with malware. The emergence of autonomous AI coding agents introduces a further qualitative shift in risk, as recent critical vulnerabilities in tools such as Cursor and Claude Code have demonstrated.

The LiteLLM gateway and Pangea AI Guard combination addresses these risks through an architectural approach: routing all LLM traffic through a central proxy, eliminating direct developer access to provider API keys, and applying real-time content scanning to both prompts and responses to detect and redact secrets, PII, and sensitive code before it reaches an external model. These are just 2 combining to address the challenge, but there are many more, offering different focus, and these are listed.

The central recommendation of this document is that voluntary developer behaviour is not a viable control at scale. Security must be enforced architecturally – through mandatory LLM gateway routing, content scanning on the wire, sensitivity-based model routing, pre-commit secret scanning, automated SAST and SCA tooling in CI/CD pipelines, and comprehensive audit logging. The regulatory environment, including the EU AI Act, OWASP LLM Top 10, and NIST AI RMF, is converging on these controls as baseline expectations.

Organisations that invest now in layered, architectural AI security controls will be positioned to adopt emerging agentic AI capabilities safely. Those that do not should expect escalating incidents as the threat landscape matures through 2027 and beyond.

2 The Real and Emerging Risks

The adoption of LLM-powered coding assistants has become near-universal. Stack Overflow's 2024 Developer Survey found 84% of developers use or plan to use AI tools in their workflows, and CrowdStrike reports that up to 90% of developers were using these tools by 2025.

While original risks were identified related to aspects like halucination or personal keys being incorporated for training, and appearing in subsequent code generated, things have improved through awareness and better practices. However, carelesness can easily create critical risk situations still, and the dependency on LLMs in development increases this liklihood.

Figure 1 - Security Risks are still around !!!

This explosive adoption has outpaced the security controls around it, creating a substantial and growing attack surface.

2.1 Secrets and Credentials Leaking into Training Data

This is perhaps the most concrete and alarming risk. In early 2025, Truffle Security scanned 400 terabytes of Common Crawl's December 2024 archive — a dataset widely used to train LLMs from OpenAI, Google, Meta, DeepSeek, and others. They discovered 11,908 live, valid secrets (API keys, passwords, tokens) embedded in 2.67 billion web pages. These included AWS root keys, Slack webhooks, Mailchimp API keys, and more. 63% of the discovered secrets were reused across multiple web pages; one WalkScore API key appeared 57,029 times across 1,871 subdomains.

The implication is direct: LLMs trained on this data learn to reproduce insecure patterns. They suggest hardcoded credentials because that's what they've seen millions of times. The problem is circular — developers use AI assistants that generate code with embedded secrets, that code gets pushed to public repositories, which then feeds the next generation of training data.

2.2 Code Sent to LLM Providers Becomes an Exposure Surface

When developers use cloud-hosted AI assistants, code snippets, configuration files, environment variables, and context are transmitted to third-party servers. Even where providers commit to not using customer data for training (as with enterprise tiers of Copilot, Claude, etc.), the data is still in transit, processed in memory, and potentially logged. The risks include: proprietary source code reaching external servers, API keys and database credentials included in prompts for debugging, internal architecture details revealed through code context, and business logic exposure that could benefit competitors. While it may not be used for training, it is leaving your network, and out of your control.

2.3 Vulnerable Code Generation

Academic research consistently shows that AI-generated code carries a high rate of security flaws. A 2025 Veracode study found that **45% of AI-generated code contains classic vulnerabilities** from the OWASP Top 10, with little improvement over two years despite newer and larger models. A study of GitHub Copilot found that 43.88% of code strings contained Common Weakness Enumerations (CWEs), and 70% of AI-assisted projects had at least one medium-to-high severity vulnerability. The CWEval benchmark shows a 25–35% gap between code that is functionally correct versus code that is both functionally correct and secure.

2.4 Supply Chain Poisoning via Hallucinated Dependencies

LLMs frequently hallucinate non-existent package names. Attackers have begun registering these phantom packages on npm, PyPI, and other registries, pre-loaded with malware. When a developer follows the LLM's suggestion and installs the package, they unknowingly introduce malicious code into their supply chain. This "slopsquatting" attack vector is novel and growing rapidly.

2.5 Agent and Tool Vulnerabilities

As development environments integrate ever-deeper AI agent capabilities (MCP servers, autonomous coding agents), new attack vectors have emerged. In 2025 alone: the CurXecute vulnerability (CVE-2025-54135) allowed attackers to execute arbitrary commands via Cursor's MCP integration; a Claude Code vulnerability (CVE-2025-55284) enabled data exfiltration through DNS requests via prompt injection; a Windsurf prompt injection stored malicious instructions in long-term memory, allowing persistent data

theft; and Replit's autonomous agent deleted production databases because it lacked environment separation. Tools are improving, but software always has bugs, and one hopes they are not security risks, which is, unfortunately, not always the case.

2.6 Membership Inference and Model Inversion

Attackers can query LLM APIs to determine whether specific data was in the training set (membership inference) or attempt to reconstruct approximate training data from model outputs (model inversion). If proprietary code was used in fine-tuning, fragments could potentially be extracted by determined adversaries.

3 The LiteLLM + Pangea Approach

While proxies, content scanners, firewalls, etc. have have evolved to handle typical web content and protection data in companies, the explosion of LLMs and their adoption in critical services has caused many challenges. This has not been simplified by the number of LLMs out there, providers, tools and variation in models, coming from all corners of the world.

However, even in this short time, there have been some positive developments. The combination of LiteLLM as an LLM gateway proxy with Pangea AI Guard as a content scanning and guardrail layer represents one of the more mature architectural approaches to controlling the risks described above. Understanding what each component does and how they integrate is essential for evaluating this and alternative solutions.

3.1 LiteLLM: The Gateway Layer

LiteLLM is an open-source proxy that sits between developer tools and LLM providers. It provides a single OpenAI-compatible API endpoint that routes to 100+ providers (OpenAI, Anthropic, AWS Bedrock, Google Vertex, Azure, and others). Its security-relevant capabilities include: virtual API key management (developers never see actual provider keys), per-user and per-team budget controls and rate limiting, comprehensive request/response logging with audit trails, dynamic routing (e.g., sending sensitive queries to a private local model and generic queries to a cloud provider), a guardrails framework for integrating content inspection middleware, and model whitelisting to prevent use of unapproved models.

Critically, LiteLLM eliminates the problem of developers individually managing provider API keys. Instead of sharing an OpenAI key across the team (which inevitably gets

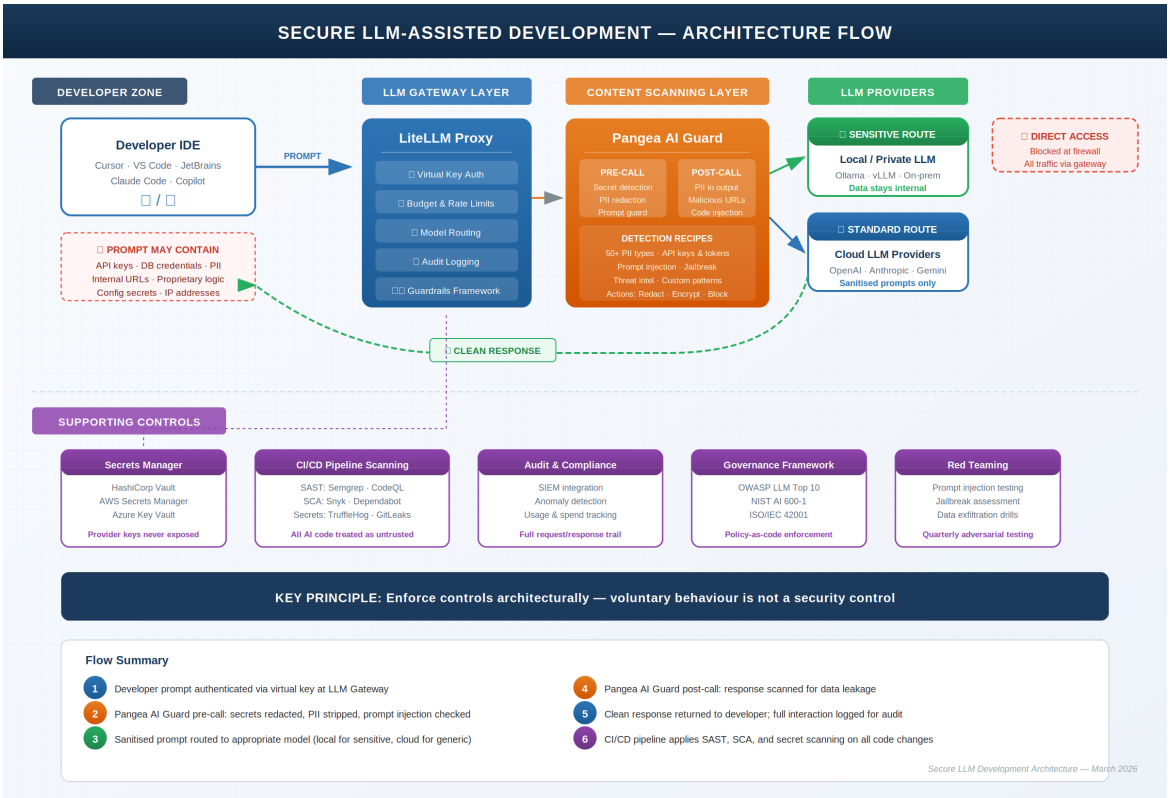
committed to GitHub), each developer gets a scoped virtual key that maps to centrally managed credentials stored in a secrets manager like HashiCorp Vault or AWS Secrets Manager.

3.2 Pangea AI Guard: The Content Scanning Layer

Pangea AI Guard integrates with LiteLLM's guardrails framework to inspect both prompts (inputs) and responses (outputs) in real time. It uses configurable detection recipes that combine over a dozen detection technologies covering: Personally Identifiable Information (PII) detection and redaction across 50+ entity types (social security numbers (SSN), credit cards, email addresses, etc.), secret and API key detection (AWS keys, tokens, encryption keys), prompt injection and jailbreak detection, malicious entity detection (dangerous URLs, IPs, domains) using threat intelligence from CrowdStrike, DomainTools, and ReversingLabs, code injection detection, content moderation for toxic or inappropriate language, and custom entity detection for organisation-specific patterns.

When Pangea detects sensitive content, it can take several actions: replace the content with a placeholder (e.g., a SSN becomes <US_SSN>), encrypt the content using format-preserving encryption (FPE) which maintains the data structure while making it unreadable, block the entire request if a critical threshold is breached, or report and log the detection for audit purposes without modifying the content.

3.3 How They Work Together



In a typical deployment, a developer's IDE (Cursor, VS Code with Copilot, etc.) is configured to point at the LiteLLM proxy instead of directly at an LLM provider. When the developer sends a prompt: LiteLLM authenticates the request using the developer's virtual key. The Pangea pre-call guardrail inspects the prompt, detects and redacts any secrets, PII, or sensitive configuration data, and checks for prompt injection. LiteLLM routes the sanitised prompt to the appropriate model (cloud or local). The response comes back through the Pangea post-call guardrail, which scans for any sensitive data in the generated output. The clean response is returned to the developer. All interactions are logged in an audit trail for compliance and incident response.

There are clear benefits with LiteLLM, and combining it with Pangea introduces enterprise level security. However, of course there are alternatives focusing more on specific aspects of the solution, from cost to performance to cloud ecosystem.

3.4 Alternative and Complementary Solutions

3.4.1 Alternative LLM Gateways

Gateway	Strengths	Considerations
Bifrost (Maxim AI)	50x faster than LiteLLM (<11µs overhead). Written in Go for production-grade performance.	Younger ecosystem, fewer integrations than LiteLLM.
Kong AI Gateway	Enterprise API management with MCP support, advanced governance, and audit capabilities.	Heavier footprint, enterprise pricing. Better suited to large organisations.
Cloudflare AI Gateway	Edge-optimised, integrates with Cloudflare's security stack, built-in caching.	Tied to Cloudflare ecosystem. Less flexible for multi-cloud.
Portkey	Developer-focused with rich observability, key management, and caching.	Less mature enterprise security features.
Pomerium + Gateway	Identity-aware proxy adding zero-trust access control to any LLM gateway.	Requires pairing with a separate gateway for LLM-specific features.

3.4.2 Alternative Content Scanning and Data Lost Prevention (DLP) Solutions

Solution	Approach
Microsoft Purview	Enterprise DLP integrated with Azure and Microsoft 365. Uses classification metadata as enforcement signals. Supports sensitivity labels that can trigger DLP rules for AI interactions.
Google Cloud DLP	API-based sensitive data detection and redaction. Integrates with Vertex AI for prompt/response scanning.
Nightfall AI	AI-native DLP specifically designed for cloud and AI workflows. Scans data in motion across SaaS, cloud, and LLM interactions.

Knostic / Kirin	AI-native security platform that secures coding assistants directly, preventing unsafe suggestions, secret leakage, and misuse of connected tools at the IDE level.
Checkmarx AI Security	Extends traditional SAST/DAST to AI-generated code. Real-time IDE scanning, governance controls, and can identify hallucinated malicious packages.

3.4.3 Complementary Controls

Pre-commit secret scanning with tools like TruffleHog, GitLeaks, or GitHub's built-in secret scanning provides a safety net for credentials that escape gateway-level controls.

Static analysis (SAST) tools such as Semgrep, SonarCloud, CodeQL, and Snyk can catch vulnerability patterns in AI-generated code before they reach production.

Software Composition Analysis (SCA) tools verify that AI-suggested dependencies are legitimate, up to date, and free of known CVEs.

OpenSSF's Security-Focused Guide for AI Code Assistant Instructions provides a framework for embedding security rules directly into AI assistant configurations (Cursor Rules, Copilot Instructions, Claude CLAUDE.md files).

4 Best Practice: Controls, Not Voluntary Behaviour

The critical insight for any engineering leader is that voluntary behaviour – telling developers to "be careful" with AI tools – does not work at scale. Controls must be architectural, enforced, and auditable. The following framework organises controls by layer.

4.1 Network and Gateway Layer (Enforced)

Mandatory LLM proxy: All LLM traffic must route through a gateway (LiteLLM or equivalent). Block direct access to LLM provider APIs at the network/firewall level. This is the single most important control – it makes all other controls possible.

Content scanning on the wire: Integrate Pangea AI Guard or equivalent as mandatory pre/post-call middleware. Configure recipes to redact secrets, PII, and proprietary identifiers before prompts reach any external model.

Sensitivity-based routing: Use the gateway to route sensitive workloads to self-hosted models (via Ollama, vLLM, or similar) and only send non-sensitive queries to cloud providers. Implement this as policy, not as developer choice.

Virtual key management: Issue per-developer or per-team virtual keys with budget limits and rate controls. Store actual provider credentials in a secrets manager (Vault, AWS Secrets Manager). Never distribute provider API keys directly.

4.2 Development Environment Layer (Enforced)

Approved tools only: Maintain an explicit allowlist of permitted AI coding assistants. Use endpoint management (MDM) or network controls to block unapproved tools.

Enterprise subscriptions: Always use enterprise tiers of AI tools which contractually guarantee that code is not used for training and provide administrative controls.

Security-first AI instructions: Adopt the OpenSSF framework for embedding security rules into AI assistant configurations. These should be committed to repositories and enforced as part of project setup, not left to individual developers.

Pre-commit hooks: Implement TruffleHog or GitLeaks as mandatory pre-commit hooks that block commits containing secrets, regardless of whether the code was AI-generated or hand-written.

4.3 CI/CD Pipeline Layer (Enforced)

Automated security scanning: Run SAST (Semgrep, CodeQL), SCA (Snyk, Dependabot), and DAST tools on every pull request. Treat all AI-generated code as untrusted by default.

Dependency verification: Validate all new dependencies against known package registries. Flag any packages that were recently created or have minimal downloads – potential "slopsquatting" attacks.

SBOM generation: Generate and maintain Software Bills of Materials with SLSA provenance attestations. This is becoming a regulatory requirement in many jurisdictions.

4.4 Governance and Audit Layer (Enforced)

Comprehensive logging: Log all LLM interactions (with content redacted/hashed as appropriate) in a SIEM. Implement anomaly detection for unusual patterns such as bulk data exfiltration attempts or prompt injection attacks.

Policy-as-code: Codify AI usage policies using OWASP's LLM Top 10 as a baseline. Map controls to NIST SP 800-218 for SDLC and ISO/IEC 42001 for AI management systems.

Regular red-teaming: Conduct periodic adversarial testing of your AI development infrastructure. A 2025 study found attack success rates of 87.2% against GPT-4 and 82.5% against Claude 2 using multi-turn jailbreak strategies. While things have improved … the risks are still there!

Mandatory review for AI-generated code: Require human review of all AI-generated code touching sensitive systems (authentication, payments, PII handling). Only 20% of developers currently use SAST tools – this must be addressed through automation, not expectation.

5 Statistics, Industry Assessment, & Outlook

5.1 Key Statistics (2024–2026)

45%
of AI-generated code contains classic OWASP Top 10 vulnerabilities, with no measurable improvement across newer or larger models (Veracode, 2025 GenAI Code Security Report)

11,908
live, valid API keys and passwords found in Common Crawl training data by Truffle Security (2025), with 63% reused across multiple pages

90%
of developers were using AI coding tools by 2025 (CrowdStrike), while 46% said they don't trust the accuracy of AI-generated results (Stack Overflow 2025, up from 31% in 2024)

Only 20%
of developers use SAST tools to scan code (2024 empirical study), meaning the vast majority of AI-generated code receives no automated security checking

25–35%
absolute gap between functionally correct and simultaneously secure code across major LLMs (CWEval benchmark, 2025). GPT-4o achieves 90.7% functional correctness but only 65.3% secure-and-functional

5.2 Regulatory and Standards Landscape

The regulatory environment is tightening rapidly. The EU AI Act is now in force and imposes obligations around transparency, risk assessment, and data governance for AI systems. OWASP's Top 10 for LLM Applications (2025 update) provides a community-driven risk taxonomy that is becoming a de facto compliance baseline. NIST's AI Risk Management Framework (AI 600-1) and the evolving generative AI profile provide US-oriented guidance. ISO/IEC 42001 establishes requirements for AI management systems, including data lineage and classification as baseline controls. The Cloud Security Alliance (CSA) has published detailed guidance on AI prompt guardrails and DLP integration as foundational controls.

5.3 How the Threat Landscape Is Evolving

5.3.1 Near-term (2026–2027)

Agentic AI introduces the most significant near-term risk amplification. As AI coding agents gain the ability to execute code, manage infrastructure, and interact with external services autonomously (via MCP and similar protocols), the blast radius of a compromised or misconfigured agent grows dramatically. The move from "code suggestion" to "autonomous action" is a qualitative change in risk profile. Expect continued growth in slopsquatting attacks as LLMs hallucinate package names at scale. Supply chain security becomes inseparable from AI security.

As AI driven development takes on more and more of the workflow from requirement to production, often with minimal oversight, the opportunities for security issues increases.

5.3.2 Medium-term (2027–2029)

Cross-modal data leakage will emerge as AI systems process code, documentation, images, and voice simultaneously. Model-level attacks (data poisoning, training data extraction) will become more sophisticated. Regulatory enforcement will begin to bite, with fines and liability for organisations that cannot demonstrate adequate AI governance. Expect consolidation in the AI security tooling market, with gateway + guardrails + audit becoming a standard "AI security stack." Unfortunately there will be many burned organisations en-route to this more holisitic, and hopefully better, approach.

5.3.3 Long-term considerations

The fundamental tension between AI capability and security will persist. This has always been the challenge as technology increases in capability, security is not given the attention it deserves until something happens. Organisations that invest now in architectural controls (gateway, scanning, audit) will be able to adopt new AI capabilities quickly and safely. Those relying on voluntary behaviour or tool-level promises will face escalating incidents. The organisations that will succeed are those that treat AI-generated code as untrusted by default and build layered verification accordingly.

6 Summary Recommendations

#	Action	Control Type	Priority
1	Deploy an LLM gateway (LiteLLM or equivalent) and block direct provider access at the network level	Architectural	Critical
2	Integrate content scanning (Pangea AI Guard or equivalent) as mandatory pre/post-call middleware	Architectural	Critical
3	Implement sensitivity-based routing: local models for sensitive code, cloud for generic queries	Architectural	High
4	Deploy pre-commit secret scanning (TruffleHog/GitLeaks) as mandatory hooks	Automated	Critical
5	Run SAST and SCA on all pull requests; treat AI-generated code as untrusted	Automated	High
6	Adopt OpenSSF security instructions in all AI assistant configurations	Standardised	High
7	Use enterprise-tier AI subscriptions with contractual no-training guarantees	Contractual	High
8	Conduct quarterly red-teaming of AI development infrastructure	Operational	Medium
9	Map controls to OWASP LLM Top 10, NIST AI 600-1, and ISO 42001	Governance	Medium
10	Build developer training programme covering secure prompting and AI risk awareness	Educational	Medium

Key Principle

The most effective security posture treats AI-generated code as untrusted by default, enforces controls architecturally rather than through developer discipline, and maintains comprehensive audit trails. Voluntary behaviour is not a security control.

7 Appendix A: References

All statistics, claims, and data points cited in this report are traceable to the following sources. References are grouped by the report section in which they are primarily cited. For a more holistic view around the risks, I recommend you dig into some of them.

7.1 Section 1: The Real and Emerging Risks

[1] *Truffle Security Co. (2025).* "Research Finds 12,000 'Live' API Keys and Passwords in DeepSeek's Training Data." Blog post, February 2025. Scanned 400TB of Common Crawl December 2024 archive; found 11,908 live secrets across 219 types; 63% reuse rate. https://trufflesecurity.com/blog/research-finds-12-000-live-api-keys-and-passwords-in-deepseek-s-training-data

[2] *Veracode (2025).* "2025 GenAI Code Security Report." Published July 2025. Analysed 80 coding tasks across 100+ LLMs; found 45% of AI-generated code introduces OWASP Top 10 vulnerabilities; security performance flat despite model improvements. https://www.veracode.com/resources/analyst-reports/2025-genai-code-security-report/

[3] *Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2025).* "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions." Communications of the ACM. Found 43.88% of Copilot code strings contained CWEs; 70% of AI-assisted projects had at least one medium-to-high severity vulnerability.

[4] *Peng, J., Cui, L., Huang, K., Yang, J., & Ray, B. (2025).* "CWEval: Outcome-driven Evaluation on Functionality and Security of LLM Code Generation." IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code), pp. 33–40. Reported 25–35% gap between functional correctness and secure-and-functional code; GPT-4o: func@10 = 90.7%, func-sec@10 = 65.3%. https://arxiv.org/abs/2501.08200

[5] *Kaspersky (2025).* "Security Risks of Vibe Coding and LLM Assistants for Developers." Blog post, October 2025. Documents CurXecute (CVE-2025-54135), Claude Code (CVE-2025-55284), Windsurf long-term memory injection, and Replit database deletion incidents. https://www.kaspersky.com/blog/vibe-coding-2025-risks/54584/

[6] *CrowdStrike (2025).* "Security Flaws in DeepSeek-Generated Code." Blog post, December 2025. Reports up to 90% of developers using AI coding tools by 2025. https://www.crowdstrike.com/en-us/blog/crowdstrike-researchers-identify-hidden-vulnerabilities-ai-coded-software/

[7] *Endor Labs (2025).* "The Most Common Security Vulnerabilities in AI-Generated Code." Blog post, August 2025. Covers dependency overuse in AI-generated code and

temporal gaps in training data leading to re-introduction of patched CVEs. https://www.endorlabs.com/learn/the-most-common-security-vulnerabilities-in-ai-generated-code

[8] *OWASP Foundation (2025).* "OWASP Top 10 for LLM Applications 2025." Defines the authoritative risk taxonomy: prompt injection, insecure output handling, training data poisoning, sensitive information disclosure, insecure plugin design, excessive agency, and overreliance. https://owasp.org/www-project-top-10-for-large-language-model-applications/

7.2 Section 2: LiteLLM + Pangea AI Guard

[9] *LiteLLM Documentation (2025).* "Data Privacy and Security." SOC 2 Type II in progress; ISO 27001 certified; describes encryption, virtual key management, and audit logging. https://docs.litellm.ai/docs/data_security

[10] *LiteLLM Documentation (2025).* "Self-Hosted Security & Encryption FAQ." Details at-rest encryption of API keys and credentials using NaCl SecretBox, transit encryption, and log handling. https://docs.litellm.ai/docs/proxy/security_encryption_faq

[11] *Pangea / CrowdStrike (2025).* "AI Guard — Overview." Describes layered detection approach: 50+ PII types, secret/key detection, prompt injection, malicious entity detection (CrowdStrike, DomainTools, ReversingLabs threat intelligence), format-preserving encryption. https://pangea.cloud/docs/ai-guard/overview

[12] *Pangea / CrowdStrike (2025).* "Integration Options — LiteLLM." Documents Pangea AI Guard integration with LiteLLM guardrails framework, including pre-call and post-call recipe configuration. https://pangea.cloud/docs/integration-options/api-gateways/litellm

[13] *NetFoundry (2025).* "Deploying a Secure, Intelligent LLM Gateway." Blog post, December 2025. Describes sensitivity-based routing architecture using LiteLLM with local embedding for classification and OpenZiti for zero-trust overlay. https://netfoundry.io/ai/deploying-a-secure-intelligent-llm-gateway/

7.3 Section 3: Alternative and Complementary Solutions

[14] *Maxim AI (2025).* "List of Top 5 LLM Gateways in 2025." Compares Bifrost (<11µs overhead, 50x faster than LiteLLM), Cloudflare AI Gateway, Kong AI Gateway, Vercel AI Gateway, and LiteLLM. https://www.getmaxim.ai/articles/list-of-top-5-llm-gateways-in-2025/

[15] *Pomerium (2025).* "LiteLLM Alternatives: Best Open-Source and Secure LLM Gateways in 2025." Reviews Portkey, Pomerium, OpenRouter, LangServe, and Semantic Kernel as alternatives. https://www.pomerium.com/blog/litellm-alternatives

[16] *Infralovers GmbH (2025).* "LiteLLM: Flexible and Secure LLM Access for Organizations." Medium, October 2025. Practical deployment guide covering prompt redaction middleware, routing, observability, and privacy controls. https://medium.com/@infralovers/litellm-flexible-and-secure-llm-access-for-organizations-4dd19720f04b

[17] *OpenSSF Best Practices Working Group (2025).* "Security-Focused Guide for AI Code Assistant Instructions." Published August 2025. Framework for Claude markdown, Copilot instructions, Cursor rules, and Kiro steering files. https://best.openssf.org/Security-Focused-Guide-for-AI-Code-Assistant-Instructions

7.4 Section 4: Best Practices and Controls

[18] *Cloud Security Alliance (2025).* "How to Build AI Prompt Guardrails: An In-Depth Guide." Published December 2025. Covers DLP as foundation for prompt guardrails, classification/labelling, online tokenisation, and governance frameworks. References NIST SP 800-207, ISO/IEC 42001, Microsoft Purview, and Google DLP. https://cloudsecurityalliance.org/blog/2025/12/10/how-to-build-ai-prompt-guardrails-an-in-depth-guide-for-securing-enterprise-genai

[19] *Knostic (2025).* "How to Secure AI Coding Assistants and Protect Your Codebase." Blog post, September 2025. Reports only 20% of developers use SAST tools (2024 empirical study of 1,003 developers); 67% of business-critical failures were change-related. https://www.knostic.ai/blog/ai-coding-assistant-security

[20] *Knostic (2025).* "Governance for your AI Coding Assistant." Blog post, October 2025. Reports 84% developer AI adoption (Stack Overflow 2025); 46% don't trust accuracy (up from 31%); 87.2% GPT-4 attack success rate in red-teaming study. https://www.knostic.ai/blog/ai-coding-assistant-governance

[21] *Checkmarx (2025).* "2025 CISO Guide to Securing AI-Generated Code." Blog post, November 2025. Covers governance frameworks, approved tool policies, real-time IDE scanning, and the finding that 29.5% of Python and 24.2% of JavaScript Copilot snippets contained security weaknesses. https://checkmarx.com/blog/ai-is-writing-your-code-whos-keeping-it-secure/

[22] *Skywork AI (2025).* "Agentic AI Safety & Guardrails: 2025 Best Practices for Enterprise." Blog post, September 2025. Covers SLSA provenance, SBOMs, workload

identity federation (SPIFFE/SPIRE), memory hygiene, and DLP on ingestion/output for agentic systems. https://skywork.ai/blog/agentic-ai-safety-best-practices-2025-enterprise/

7.5 Section 5: Statistics and Industry Assessment

[23] *Stack Overflow (2025).* "2025 Stack Overflow Developer Survey." 84% of respondents use or plan to use AI tools (up from 76% in 2024); 46% actively distrust AI output accuracy (up from ~31%); 51% of professional developers use AI daily. https://survey.stackoverflow.co/2025

[24] *Stack Overflow (2024).* "2024 Stack Overflow Developer Survey." 76% of respondents using or planning to use AI tools; 62% actively using (up from 44% in 2023); ChatGPT (82%), GitHub Copilot (41%), Google Gemini (24%) as top AI search tools. https://survey.stackoverflow.co/2024/ai

[25] *Mend.io (2025).* "LLM Security in 2025: Key Risks, Best Practices & Trends." October 2025. Practical overview of OWASP Top 10 for LLM Applications with detailed remediation strategies per category. https://www.mend.io/blog/llm-security-risks-mitigations-whats-next/

[26] *Cycode (2025).* "The 2025 OWASP Top 10: Addressing Software Supply Chain and LLM Risks." Blog post, November 2025. Maps OWASP LLM guidance to NIST SP 800-218 and recommends unified AI-native security platforms. https://cycode.com/blog/the-2025-owasp-top-10-addressing-software-supply-chain-and-llm-risks-with-cycode/

[27] *Oligo Security (2025).* "LLM Security in 2025: Risks, Examples, and Best Practices." Covers RBAC, IAM integration, behavioural rate limiting, dataset provenance tracking, and red-teaming recommendations. https://www.oligo.security/academy/llm-security-in-2025-risks-examples-and-best-practices

[28] *Cloud Security Alliance (2025).* "Understanding Security Risks in AI-Generated Code." July 2025. Reports 62% of AI-generated code contains design flaws or known vulnerabilities; covers SQL injection pattern reproduction and missing security controls in AI output. https://cloudsecurityalliance.org/blog/2025/07/09/understanding-security-risks-in-ai-generated-code

[29] *USCSI (2025).* "What are LLM Security Risks and Mitigation Plan for 2026." Covers training-time vs. inference-time risk categorisation, data poisoning, model extraction, membership inference, and model inversion attacks. https://www.uscsinstitute.org/cybersecurity-insights/blog/what-are-llm-security-risks-and-mitigation-plan-for-2026

[30] *Graphite (2025).* "Privacy and Security Considerations When Using AI Coding Tools." Practical guidance on enterprise subscriptions, secure configurations, human oversight, and data handling policies for AI coding assistants. https://graphite.com/guides/privacy-security-ai-coding-tools

[31] *DX / GetDX (2025).* "AI Code Generation: Best Practices for Enterprise Adoption in 2025." Covers governance frameworks, quality assurance integration, and the amplification effect of AI on existing development practices. https://getdx.com/blog/ai-code-enterprise-adoption/

[32] *StackHawk (2025).* "AI Code Security: 4 Best Practices Every Developer Should Know." September 2025. Covers Cursor Rules configuration for security, healthcare/HIPAA-specific AI rules, and the need for DAST integration. References Stack Overflow 2024 data (76% AI tool usage). https://www.stackhawk.com/blog/4-best-practices-for-ai-code-security-a-developers-guide/