InnovAIte SLOVAKIA

InnovAIte Slovakia Deliverable

D6.3: Report on submitting sensitive source code and data to AI models

WP6: Automating Software Development with AI: Enhancing Efficiency

Version: **00.10~~1~~**
Date: July ~~19~~7, 2026

Funded by the European Union NextGenerationEU
RECOVERY AND RESILIENCE PLAN
DEPUTY PRIME MINISTER'S OFFICE FOR THE RECOVERY PLAN AND THE KNOWLEDGE ECONOMY
VAIA RESEARCH AND INNOVATION AUTHORITY

1 Abstract

Critical findings from empirical research (2025–2026):

Finding	Source	Severity
99.4% jailbreak success rate on GitHub Copilot *(requires attacker-controlled content in developer's editing context — e.g., malicious code file or repository content)*	AAAI-25 (Security Attacks on LCCTs)	Critical
Zero-click data exfiltration in production (EchoLeak)	CVE-2025-32711	Critical (CVSS 9.8)
RoguePilot: GitHub Copilot repository takeover via passive prompt injection	Orca Security, Feb 2026	Critical (CVSS 9.6)
Command injection via ModelScope ms-agent misconfiguration	CVE-2026-2256, Mar 2026	Medium (CVSS 6.5)
AI/LLMs ranked #1 cybersecurity concern (29%), surpassing ransomware (21%)	Arctic Wolf 2025 Trends Report	Industry-wide
87% of leaders identify AI vulnerabilities as fastest-growing cyber risk	WEF Cybersecurity Outlook 2026	Industry-wide

The report evaluates four primary mitigation paradigms — **Prevention/Isolation**, **Obfuscation**, **Secure Channel**, and **Thick Client (Local Deployment)** — against the current threat landscape. The analysis reveals that no single approach is universally optimal. The emerging best practice is a **hybrid architecture** combining local deployment for sensitive workloads with governed cloud access for non-sensitive tasks, underpinned by inline AI runtime security and comprehensive governance frameworks.

The **EU AI Act's August 2, 2026 deadline** for high-risk AI system compliance creates urgent regulatory pressure, with penalties reaching 7% of global annual turnover for violations. Organisations must act now to establish governance structures, technical documentation, and human oversight mechanisms.

This report's analysis leads to the following conclusions:

- **No single approach is sufficient.** The four paradigms (Prevention, Obfuscation, Secure Channel, Thick Client) each address different aspects of the threat landscape.
- **The Thick Client approach has become economically viable.**
- **The hybrid architecture is the emerging best practice.**
- **Regulatory compliance is now a forcing function.**

- **FHE represents the long-term solution.** While not yet production-ready for large LLMs.

Treat AI security as a first-class security domain.

- Implement the hybrid architecture.
- Address Shadow AI immediately.
- Prepare for the EU AI Act deadline.
- Monitor the FHE research trajectory.
- Treat agentic AI as a new threat category.

---

2 Introduction: The AI-Assisted Development Paradox

2.1 The Fundamental Shift in Software Engineering

The contemporary software engineering landscape is defined by a fundamental shift from deterministic manual coding to probabilistic, AI-assisted synthesis. Large Language Models (LLMs), characterised by transformer architectures and self-attention mechanisms, have demonstrated a significant capacity to refactor, debug, and generate complex source code. For a modern enterprise-scale software house, the integration of LLMs into the Software Development Life Cycle (SDLC) offers measurable reductions in technical debt and increases in developer velocity. Organisations that have successfully integrated these models report productivity gains in routine tasks such as unit test generation, documentation, and boilerplate reduction.

However, this rapidly expanding industrial adoption of AI-assisted development faces a critical inflection point regarding the security of sensitive resources. The paradox lies in the data-hungry nature of these models: to provide contextually relevant assistance, an LLM typically requires access to the surrounding codebase, architectural diagrams, and internal documentation. In the prevailing "AI-as-a-Service" (AIaaS) model, this data is transmitted as a series of tokens over public or semi-public networks to remote inference servers.

2.2 The Trust Gap

This telemetry poses a **multidimensional risk** that spans the network, the model provider's infrastructure, and the developer's own toolchain:

- **Network interception** through Man-in-the-Middle (MitM) attacks — even TLS-encrypted connections can be intercepted by corporate proxies, misconfigured certificates, or adversary-in-the-middle tooling targeting developer workstations.

- **Data residualization** within the model's weights during subsequent training iterations — if the provider uses submitted prompts for fine-tuning (absent a Zero-Data Retention agreement), proprietary code patterns may become recoverable by other users through targeted prompt engineering.
- **Exposure of proprietary logic** to third-party providers — the provider's employees, subprocessors, and legal jurisdiction all gain potential access to submitted code, creating contractual, regulatory, and competitive risks.
- **AI-mediated supply chain attacks** — a new threat class where LLMs automatically execute malicious instructions embedded in developer content (e.g., code comments, README files, issue descriptions), bypassing human review entirely and enabling attackers to weaponize the AI assistant itself against the developer.

The submission of sensitive source code — often protected by Non-Disclosure Agreements (NDAs) or classified as core intellectual property (IP) — to an external AI endpoint creates a "Trust Gap" that traditional perimeter security is ill-equipped to close. This gap has widened dramatically in 2025–2026 as coding assistants have evolved from simple autocomplete tools into fully autonomous agentic systems capable of reading files, executing commands, browsing the web, and committing code to repositories.

2.3 Scope and Methodology

This report addresses security issues related to the following reference architecture, as defined in the D6.3 deliverable specification:

- **Developer → IDE/Coding Agent → LLM Proxy → Cloud or Local LLM**
- **Developer Machine → MCP (Model Context Protocol) → Tools, File System, Environment, Executables**
- **Integration with Git, CI/CD, Issue Tracking, and external web resources**

2.3.1 Literature Review Scope and Cutoff

The analysis is grounded in 91 references spanning peer-reviewed research, industry standards, vulnerability advisories, and industry reports. The systematic literature review was conducted as follows:

- **Literature review cutoff date**: **2026-03-21** (date of final source material inclusion). Developments after this date (e.g., the EU Parliament vote on AI Act implementation delays on 2026-03-26) are noted where relevant but are not systematically incorporated.
- **Source selection methodology**: Sources were identified through (1) structured search of academic databases (IEEE Xplore, ACM Digital Library, arXiv, IACR ePrint) using keywords

related to LLM security, code submission risks, and agentic AI; (2) monitoring of industry security advisories (OWASP, NIST, CVE databases); and (3) direct project experience with the tools and architectures described in §16 (Lessons Learned). Priority was given to sources with empirical evidence over purely theoretical treatments.

- **Regulatory freeze date**: The regulatory analysis (§10) is frozen as of **2026-03-21**. Given the rapid pace of regulatory evolution (particularly regarding the EU AI Act), readers should verify current regulatory status against official sources.

Post-cutoff additions During peer-review remediation (June 2026), a small number of cited sources were added whose *publication* post-dates the 21 March 2026 cutoff. These are: the EU Parliament AI Act vote of 26 March 2026 [50] (already noted in §10.1); the Cloud Security Alliance report of 25 March 2026 [89] (§7.11); the Stanford HAI *AI Index 2026* [75] (published 13 April 2026, reporting data as of March 2026; §8.5.8); the SnackonAI SWE-bench analysis of 5 May 2026 [10] (§7.11); and the arXiv preprint "Coding Benchmarks Are Misaligned" [85] (June 2026; §7.11). Each is flagged in-text at its citation site. No post-cutoff addition changes the report's core findings or recommendations; they extend coverage of an evolving landscape. Readers should re-verify current regulatory and threat-landscape status against primary sources.

2.3.2 Source Type Classification

References in this report are classified by evidence quality as follows:

Category	Label	Description	Examples in this report
A.1	[Peer-Reviewed]	Published at peer-reviewed conferences or in peer-reviewed journals	AAAI-25, ICML 2025, NDSS 2025, ACM CCS 2025, EACL 2026, *Information Fusion*, *AI Open*
A.2	[Pre-print]	arXiv, IACR ePrint, HAL, or conference submissions not yet accepted	arXiv:2506.19676, ePrint 2026/105
A.3	[Technical Resource]	Non-publication technical resources (websites, project documentation)	TEE.fail
B	[Standard/Framework]	Official industry standards and frameworks	OWASP, NIST, NCSC/CISA

Category	Label	Description	Examples in this report
C	[Advisory]	Vulnerability advisories and incident reports	CVE entries, vendor security advisories
D	[Industry Report]	Vendor surveys, analyst reports, blog analyses	Gartner, WEF, Dark Reading

Methodological caveat: A significant portion of the evidence base (§A.2, §C, §D) consists of pre-prints, vendor reports, and blog analyses rather than formally peer-reviewed research. This reflects the nascent and rapidly evolving nature of the field, where the lag between incident and publication can exceed 12 months. Where empirical claims rely primarily on non-peer-reviewed sources, this is noted in the text. Readers should exercise appropriate caution when citing such findings as established facts.

2.4 The Evolving Threat Context

The threat landscape has evolved dramatically since the initial conception of this deliverable. What were once theoretical risks have become actively exploited vulnerabilities:

- **May 2025:** EchoLeak (CVE-2025-32711) — first real-world zero-click prompt injection exploit in a production LLM system (Microsoft 365 Copilot) [2]
- **June 2025:** CVE-2025-55284 — Claude Code DNS exfiltration of API keys and secrets
- **February 2026:** RoguePilot — GitHub Copilot repository takeover via passive prompt injection in repository content
- **March 2026:** CVE-2026-2256 — Command injection in ModelScope ms-agent via excessive tool permissions [20a], [20]
- **March 2026:** Web-based indirect prompt injection observed in the wild against production AI agents (Palo Alto Unit 42)

Dark Reading's 2026 reader survey identified **agentic AI attacks** as the most likely trending security reality for 2026. The MCP ecosystem alone has reportedly grown to **97 million monthly SDK downloads** and **10,000+ active tool servers**, creating an enormous and largely unsecured attack surface. *(industry estimate; these figures could not be independently verified against primary MCP registry data)*

3 Scientific Foundations: Privacy-Preserving AI Research

3.1 The Privacy-Utility Paradox

Scientific efforts in privacy-preserving AI have focused heavily on the **"Privacy-Utility Paradox"** — the fundamental tension between maximizing the helpfulness of LLM responses while minimising the leakage of sensitive training or prompt data. This paradox is not merely theoretical; it has direct operational consequences for organisations deploying AI-assisted development tools.

The core challenge is that the same contextual richness that makes an LLM useful for code generation — its exposure to large volumes of real-world code, including proprietary implementations — is precisely what creates privacy risks. A model trained on a company's internal codebase may inadvertently reproduce proprietary algorithms when queried by unauthorised parties.

3.2 Differential Privacy for Code Models

One of the most promising research streams is **Differential Privacy (DP)**. By applying DP to the fine-tuning of code-centric LLMs, researchers aim to ensure that the inclusion of a specific proprietary code snippet in the training set does not significantly alter the probability of the model generating that exact snippet for an unauthorised user.

Mathematical frameworks like (ϵ, δ)**-differential privacy** are now being adapted to the discrete token space of programming languages. The paradigm is based on adding calibrated noise to computations so no individual training example can be reverse-engineered from the output.

Critical limitation: In practice, adding enough noise to protect the privacy of training code makes the generated code too broken to be useful, especially for complex algorithms where there is essentially only one correct implementation. This "utility cliff" remains a significant unsolved problem in the field.

Implication for organisations: Differential Privacy cannot be relied upon as a practical protection for code models today. Organisations should not assume DP provides meaningful privacy guarantees for proprietary training data — the utility trade-off is currently prohibitive for production code generation systems.

3.3 Fully Homomorphic Encryption (FHE): From Theory to Emerging Practice

Fully Homomorphic Encryption (FHE) represents the theoretical "gold standard" for privacy-preserving inference: the LLM processes code in its encrypted state, returning an encrypted

suggestion that the developer decrypts locally. This effectively nullifies the network transfer risk, as the provider never has access to the plaintext logic.

2025–2026 Breakthrough Research:

Publication	Achievement	Institution	Date
EncryptedLLM (ICML 2025)	200x GPU speedup over CPU-based FHE	Carnegie Mellon University	Oct 2025
Hyperion (ePrint IACR 2025/2318)	100x latency improvement for token sampling; 0.14s for 32k vocabulary	UC Santa Barbara	Dec 2025
ELLMo (ePrint IACR 2026/198)	Packing- and depth-aware optimisation for encrypted transformer inference	Boston University	Feb 2026
Orion (ASPLOS 2025 Best Paper)	Breakthrough in FHE for deep learning	NYU Tandon	Apr 2025

Sources: [6] EncryptedLLM; [7] Hyperion; [8] ELLMo; Orion (ASPLOS 2025 Best Paper).

Practicality Assessment (March 2026):

Model Size	FHE Status	Timeline to Production
Small (<1B params)	Emerging practical	Now – 6 months
Medium (1–10B params)	Research/prototype	6–18 months
Large (>10B params)	Not viable	18–36 months
Token sampling	Solved (Hyperion)	Now

Provenance note: Timeline projections synthesised from [6], [7], [8] and ePrint IACR 2026/105 [45]; these represent the authors' interpretation of current research trajectories based on the cited papers, not independently validated empirical predictions.

Conclusion: FHE is transitioning from "theoretically possible" to "practically emerging" but remains **not production-ready for large LLMs** as of March 2026. The long-term outlook is promising:

"FHE represents the natural asymptotic endpoint of privacy-preserving inference trajectory: inference directly over ciphertexts, without enclaves or trusted hardware, with confidentiality guaranteed purely by cryptography." — *Privacy-Preserving LLM Inference in Practice* (ePrint IACR 2026/105)

3.4 Trusted Execution Environments (TEEs)

Trusted Execution Environments (TEEs) — including Intel SGX, AMD SEV, and ARM TrustZone — offer a more mature alternative to FHE for production use. TEEs create hardware-isolated enclaves where LLM inference can occur without exposing data to the host operating system or cloud provider.

Comparative Analysis: TEE vs. FHE

Criterion	TEE (SGX/SEV/TrustZone)	FHE
Performance	Near-native	10–1000x slower
Security Model	Hardware trust required	Pure cryptography
Attack Surface	Side-channel vulnerabilities	Mathematical assumptions
Production Ready	Yes (with caveats)	No (emerging)
Verification	Attestation protocols	Mathematical proof

Authors' analysis; TEE characteristics from [36]–[38], FHE characteristics from [6]–[8], [45].

Important caveat: Recent research has exposed fundamental weaknesses in TEE implementations. The TEE.Fail attack (Georgia Tech / Purdue / Synkhronix, October 2025) demonstrated that DDR5 memory bus interposition can extract secrets from Intel SGX, TDX, and AMD SEV-SNP enclaves using less than $1,000 of hobbyist-level hardware [36]. Beyond this physical attack, cache side-channel attacks show varying success rates depending on technique: Flush+Reload achieves **99.2%** AES key recovery, while Prime+Probe achieves only **53.4%** even with extensive measurements [37a]. Energy-based attacks (CLKSCREW, 2018) can extract cryptographic keys via voltage/frequency manipulation [37b]. TEEs impose **+4–10% throughput overhead** and **<20% latency overhead** for confidential LLM inference, and limit model sizes to approximately 30B parameters due to enclave memory constraints [38]. TEEs are currently the more practical option than FHE but are not without significant limitations. For maximum security, air-gapped local deployment remains superior to TEE-isolated cloud.

3.5 Membership Inference Attacks and Privacy Extraction

Membership Inference Attacks (MIAs) on code models have become a standard benchmark for measuring privacy leakage. These attacks allow an adversary to determine, with statistical confidence, whether a specific proprietary algorithm was used to train a model — effectively confirming the presence of stolen IP in a model's weights.

The AAAI-25 paper *"Security Attacks on LLM-based Code Completion Tools"* demonstrated that privacy extraction is not merely theoretical. Using code-driven privacy extraction attacks against GitHub Copilot, researchers successfully extracted:

- **2,173 real GitHub usernames** (80.36% accuracy)
- **54 exact matching email addresses**
- **314 matching locations** (100 exact + 214 fuzzy matches)

This empirical evidence confirms that training data — including proprietary code submitted to cloud LLMs — can be extracted by adversaries with sufficient query access.

Implication for organisations: If your organisation has submitted proprietary source code to a cloud LLM provider that uses submitted data for training — without a Zero-Data Retention agreement — an adversary with API access to that model may already be able to extract fragments of your code through targeted membership inference queries. This is not a future risk; the AAAI-25 study demonstrated it against a production system. The mitigation is ZDR agreements (§9.3) and local deployment (§8.5) for sensitive workloads.

3.6 Privacy-Preserving Prompt Engineering (PPPE)

An emerging practical approach is **Privacy-Preserving Prompt Engineering (PPPE)**, where sensitive context is automatically abstracted into generic symbolic representations before being dispatched to the model. Proprietary identifiers and logic flows are mapped to a symbolic dictionary before transmission. For example, a proprietary trading algorithm is transformed into a generic mathematical abstraction. Upon the model's response, the transformation is reversed.

Research from EASE 2025 [41] revealed a counter-intuitive finding: LLMs often **reduce** cyclomatic complexity when attempting obfuscation, with GPT-4-Turbo achieving an 81% pass rate with few-shot prompting versus 29% with standard prompting. This "Simplicity by Obfuscation" phenomenon has important implications for the obfuscation mitigation paradigm discussed in §8.3 (Functional Obfuscation and Semantic Masking).

Implication for organisations: If your organisation is using LLM-assisted obfuscation as a security control, the LLM may be actively reducing your protection level. Obfuscation should not be relied upon as a primary defence mechanism — it provides a false sense of security while potentially degrading code quality and maintainability.

---

4 Information Flow Architecture and Attack Surface

4.1 The Three-Layer Communication Architecture

Based on the comprehensive survey of LLM-driven AI agent communication [3], security risks in AI-assisted development are organised across three architectural layers, each with distinct

threat profiles. Understanding which layer a given attack operates at is essential for selecting the correct mitigation: L1 attacks require network controls, L2 attacks require authentication controls, and L3 attacks — the most dangerous — require semantic-level defences that traditional security tools cannot provide.

L1: Data Transmission Layer

- Protocols: HTTPS, TCP/IP, gRPC, WebSocket
- Risks: Eavesdropping, MITM attacks, replay attacks, DoS/DDoS
- Defences: TLS 1.3, certificate pinning, traffic obfuscation, rate limiting

L2: Interaction Protocol Layer

- Protocols: OAuth 2.1, OIDC/PKCE, mTLS, session management
- Risks: Identity spoofing, session manipulation, privilege escalation, registration pollution
- Defences: Multi-Factor Authentication (MFA), continuous authentication, context isolation, authority verification

L3: Semantic Interpretation Layer (HIGHEST RISK)

- Protocols: LLM-based intent understanding, reasoning, planning
- Risks: Prompt injection, jailbreak attacks, privacy leakage, memory poisoning, knowledge corruption
- Defences: Input/output filtering, intent analysis, capability boundary control, memory integrity protection

The L3 layer is categorically the most dangerous because traditional security mechanisms are **blind to semantic intent conflicts**. A firewall can block a malicious IP address; it cannot detect that a carefully crafted variable name in a code file is instructing an LLM to exfiltrate secrets.

4.2 Developer to Cloud-Based SaaS LLMs

This section describes the data flow when a developer uses a cloud-based SaaS LLM (e.g., GitHub Copilot, Amazon Q Developer, Cursor with cloud backend). Figure 1 illustrates the complete path from the developer's IDE to the remote LLM provider's API, including all intermediate network hops where data may be intercepted or logged.

Flow:

Developer IDE (VS Code, IntelliJ, Cursor)
→ TLS 1.3 Encrypted Tunnel
→ Corporate Egress / API Gateway

→ Public Internet
→ Cloud LLM Provider API (OpenAI, Google Gemini, Anthropic Claude)

Figure 1: Developer-to-Cloud SaaS LLM data flow (Section 4.2).

Data transmitted (explicit and implicit):

- Explicit prompts authored by the developer
- Implicit context window gathering: surrounding code blocks, Abstract Syntax Trees (ASTs) of active files, workspace directory structures
- Terminal stderr logs and environment variables if inadvertently captured by the client tool
- File names, import statements, and cross-file references aggregated by the coding assistant

Data received:

- Token streams representing code completions (Fill-in-the-Middle / FIM models)
- Multi-file refactoring diffs
- Vulnerability explanations and generated unit tests

Critical risk: The implicit context gathering is largely invisible to developers. A developer asking for help with a utility function may inadvertently expose the entire workspace context, including files containing API keys, database connection strings, or proprietary business logic.

4.3 Developer to On-Premise / VPC-Deployed LLMs

This section describes the equivalent data flow when the LLM is deployed on-premise or in a private VPC. The key difference from the cloud SaaS pattern is that all traffic remains within the corporate network boundary, never traversing public internet infrastructure. However, residual risks at L2 and L3 layers still apply.

Flow:

Developer IDE / Terminal
→ Internal Corporate Network (LAN/VLAN)
→ Internal Load Balancer / Ingress Controller
→ Local LLM Inference Server
(vLLM, Ollama, TGI hosting Llama 3, Mistral, CodeLlama on internal GPU clusters)

Figure 2: Developer-to-On-Premise / VPC-deployed LLM data flow (Section 4.3).

Data transmitted: Identical JSON payloads to cloud flows (over REST/gRPC), but strictly isolated within the corporate firewall, never traversing public routing infrastructure.

Data received: Inference results generated entirely on proprietary infrastructure, ensuring zero external data residency.

Residual risks: Even in on-premise deployments, risks remain at L3 (prompt injection via poisoned internal data sources) and L2 (insider threats with access to the inference server).

Implication for organisations: On-premise deployment does not eliminate all risk; insider threats and prompt injection via internal data sources remain active threats even in air-gapped environments. Organisations must implement additional controls at the L3 layer (input/output filtering, intent analysis) regardless of deployment topology.

4.4 Retrieval-Augmented Generation (RAG) Architecture

RAG architectures enhance LLM responses by retrieving relevant context from a vector database before generating a response. This section describes the data flow for a developer prompt that triggers RAG retrieval, showing how internal documentation and code are injected into the LLM context. The security implications of this architecture — particularly RAG data poisoning — are discussed following the diagram.

Flow:

Developer Prompt
→ Query Embedding Model (e.g., text-embedding-3-small)
→ ANN Search in Vector Database (Milvus, Qdrant, pgvector)
→ Retrieval of top-k relevant chunks
(internal documentation, proprietary source code, ADRs)
→ Context Injection into System Prompt
→ LLM Inference Engine
→ Contextualized Response to Developer

Figure 3: Retrieval-Augmented Generation (RAG) architecture data flow (Section 4.4).

Security implications: RAG architectures introduce a critical new attack surface: **RAG Data Poisoning** (OWASP LLM04). If an attacker compromises the internal Confluence, GitLab, or SharePoint repository used to feed the RAG Vector Database, they can inject malicious, invisible instructions. When a developer queries the system, the RAG pipeline retrieves the poisoned chunk, manipulating the LLM into generating vulnerable code — a form of **Indirect Prompt Injection** that bypasses all direct input validation.

Additionally, the RAG system often acts as a privileged service account reading from the Vector DB. Without robust authorization (AuthZ) mechanisms, a developer might use the LLM to query and retrieve context from highly classified projects they do not have direct access to, bypassing traditional Role-Based Access Control (RBAC).

4.5 Agentic Coding Assistant Architecture

The most significant architectural evolution in 2025–2026 is the transition from passive autocomplete tools to **fully autonomous agentic coding assistants** [5]. These systems — exemplified by GitHub Copilot Workspace, Cursor, Claude Code, and similar tools — operate with dramatically expanded capabilities:

Developer Instruction
→ Coding Agent (LLM + Planning Module)
→ MCP (Model Context Protocol) Server
→ Tools: File System R/W, Shell Execution, Web Browser, Git Operations
→ External APIs: GitHub, Jira, Confluence, Package Registries
→ CI/CD Pipeline Triggers

Figure 4: Agentic coding assistant architecture (Section 4.5).

Expanded attack surface in agentic systems:

- Agents can **read and write files** across the entire workspace
- Agents can **execute shell commands** with developer-level permissions
- Agents can **browse external websites** and process their content (indirect prompt injection vector)
- Agents can **commit code and open pull requests** autonomously
- Agents can **query internal databases** and external APIs

As of February 2026, enterprise AI assistants are connected to ticketing systems, source code repositories, chat platforms, and cloud dashboards — capable of opening pull requests autonomously, querying internal databases, booking services, and triggering workflows. This level of access reflects what the Cisco AI Readiness Index 2025 [46] describes as agentic AI systems that "take action by navigating tasks, workflows, and even business decisions autonomously" — with 83% of organisations planning to deploy AI agents.

4.6 External Tool and Agent Data Flows: Perimeter Leakage Vectors

§4.5 (Agentic Coding Assistant Architecture) described the *capabilities* of agentic coding assistants. This section addresses a distinct and critically underappreciated dimension: **where the data actually goes** when those agents invoke external tools. The key insight — emphasised by enterprise security practitioners — is that "external" in this context does not mean that tools run on external infrastructure. It means that **sensitive source code and data exit the organisational perimeter** as a consequence of tool invocations, often invisibly and without developer awareness.

4.6.1 The Perimeter Leakage Model

Traditional network security defines the perimeter as the boundary between the internal corporate network and the public internet. In the context of agentic AI development tools, this boundary is routinely crossed in ways that bypass conventional monitoring:

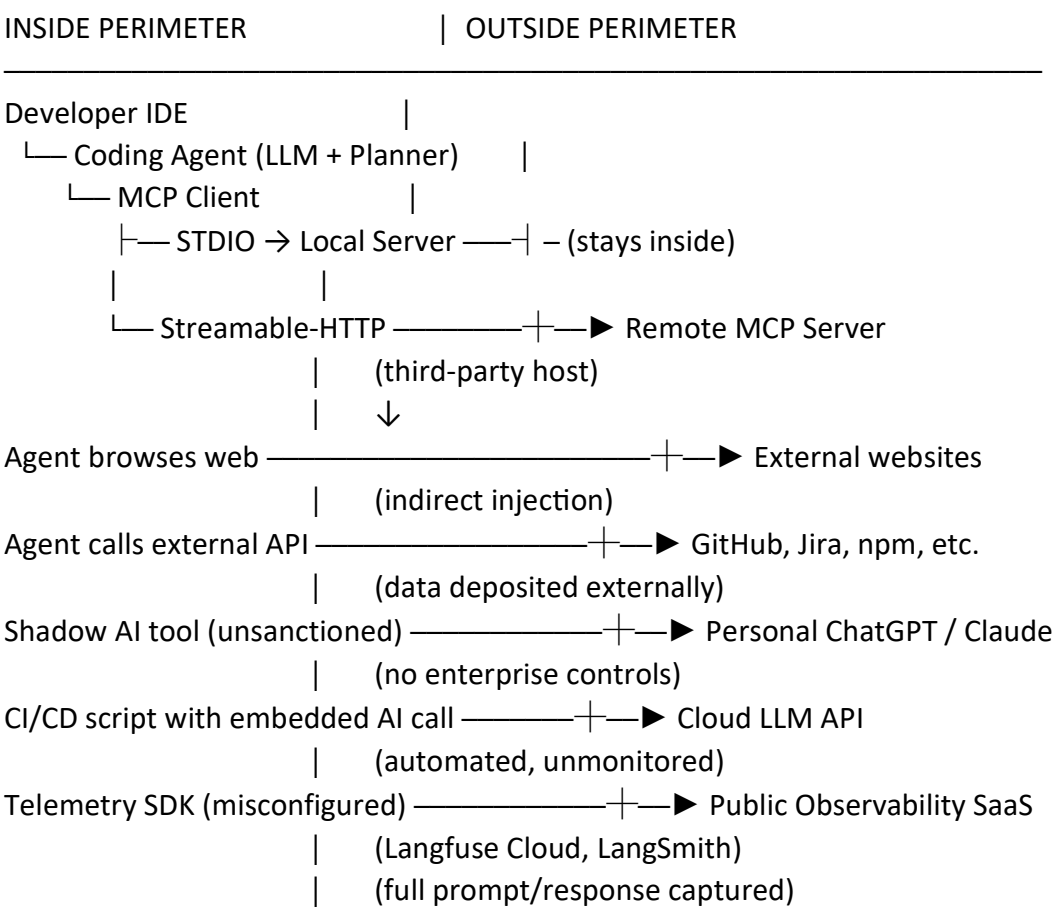


Figure 5: Perimeter leakage model (INSIDE/OUTSIDE boundary) (Section 4.6.1).

The critical distinction is that **the agent is the exfiltration vector** — not a malicious actor. The agent legitimately invokes tools as instructed, and those tool invocations carry sensitive context across the perimeter as a side effect of normal operation. A particularly insidious variant of this pattern is the **telemetry misconfiguration backdoor**: an organisation deploys a secure, on-premise LLM to keep code inside the perimeter, but connects it to a public cloud observability platform (e.g., Langfuse Cloud, LangSmith) for debugging. The telemetry SDK captures the full prompt/response stream — identical to the data the local LLM was protecting — and transmits it to the cloud, completely negating the on-premise investment.

Insecure pattern — local LLM with public cloud telemetry (data leaves perimeter):

SECURE ON-PREMISE ZONE | PUBLIC CLOUD (Third-Party SaaS)

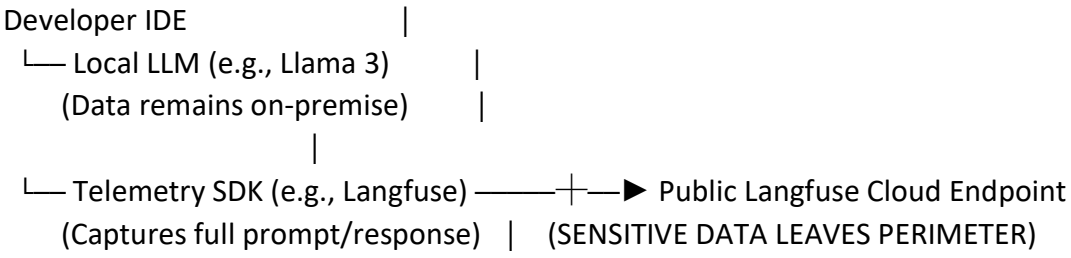


Figure 6: Insecure pattern — local LLM with public cloud telemetry (Section 4.6.1).

Secure pattern — local LLM with self-hosted telemetry (data stays on-premise):

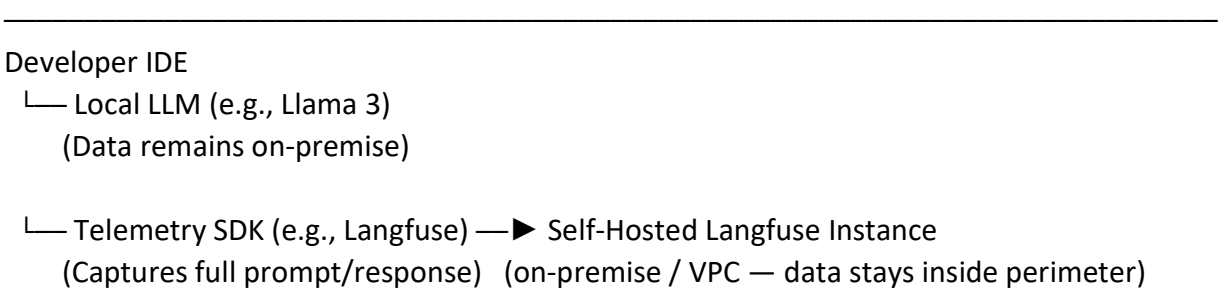


Figure 7: Secure pattern — local LLM with self-hosted telemetry (Section 4.6.1).

4.6.2 MCP Transport Modes and Their Perimeter Implications

The Model Context Protocol (MCP) supports two primary transport mechanisms, each with fundamentally different perimeter implications [15]:

STDIO (Local Transport):

- Communication occurs via subprocess stdin/stdout on the developer's machine
- Data does not cross the network perimeter during tool invocation
- **Risk profile:** Lower perimeter leakage risk, but local server may itself call external APIs
- **Monitoring:** Endpoint-level monitoring required; network monitoring insufficient

Streamable-HTTP (Remote Transport):

- Communication occurs over HTTP/S to a remote MCP server endpoint
- Every tool invocation transmits the agent's current context — which may include source code, file contents, credentials, and architectural details — to the remote server
- **Risk profile:** HIGH — data crosses the perimeter on every tool call
- **Monitoring:** Encrypted HTTPS traffic is opaque to traditional DLP systems; AI-specific proxy required

This implication applies specifically to the Streamable-HTTP (remote) transport described above — not to STDIO, which communicates locally via subprocess stdin/stdout and does not cross the network perimeter. When a developer's coding agent invokes a remote MCP tool over Streamable-HTTP (e.g., a documentation lookup, a code search service, or a package registry query), the **entire context window** — including surrounding source code, open files, and recent conversation history — may be transmitted to the remote server as part of the tool request payload. This is not a bug; it is how MCP is designed to work. The security risk arises when that context contains sensitive proprietary code.

Adoption scale of the risk: As of Q1 2026, the MCP ecosystem reportedly comprises **97 million monthly SDK downloads** and **10,000+ active tool servers** [28] *(industry estimate; the cited source is a survey article and these figures could not be independently verified against primary MCP registry data)*, the vast majority of which are third-party operated and have not undergone enterprise security review.

4.6.3 Shadow AI as a Perimeter Leakage Vector

Beyond sanctioned MCP tool usage, a parallel and largely invisible leakage channel exists through **Shadow AI** — AI tools, models, and integrations used within an enterprise without IT or security team knowledge or approval [32].

Prevalence (Q1 2026):

Metric	Value	Source
Employees using unauthorised AI tools	**68%**	Gartner [32]
Enterprises with no AI usage visibility	**80%+**	Second Talent [33]
Average additional breach cost from Shadow AI	**$670,000**	Vectra AI [34]
Annual insider risk cost from AI negligence	**$10.3M**	Vectra AI [34]
Employees continuing AI use after a ban	**~50%**	Vectra AI [34]

Shadow AI creates perimeter leakage through multiple channels that are structurally invisible to enterprise security:

- **Browser extensions:** AI writing assistants and code completers installed by developers without IT review; these extensions may transmit the active editor buffer to external endpoints
- **Personal accounts:** Developers using personal GitHub Copilot, ChatGPT, or Claude subscriptions — outside enterprise Zero-Data Retention agreements — to process proprietary code

- **CI/CD pipeline scripts:** Automated build scripts that call external AI APIs, transmitting code diffs, test results, or configuration files without human review
- **Collaboration platform bots:** Unsanctioned AI integrations in Slack or Teams channels where code snippets are routinely shared
- **Local MCP servers with external callbacks:** Developer-installed MCP servers that appear local (STDIO) but themselves make outbound calls to cloud services

Detection challenge: Shadow AI traffic is structurally indistinguishable from legitimate HTTPS traffic. Encrypted connections to api.openai.com, api.anthropic.com, or generativelanguage.googleapis.com look identical whether they originate from a sanctioned enterprise tool or a developer's personal account. Traditional DLP systems have no visibility into the payload content.

4.6.4 Tool Poisoning and Rug Pull: Adversarial Perimeter Crossing

Beyond accidental leakage, adversaries actively exploit the MCP tool ecosystem to cause deliberate data exfiltration. Two attack patterns are particularly relevant to the perimeter leakage model:

Tool Poisoning (Indirect Prompt Injection via Tool Descriptors): An attacker publishes or compromises an MCP server and embeds malicious instructions within the tool's description or parameter definitions. When the coding agent reads the tool manifest, the LLM interprets the hidden instructions and executes them — including instructions to transmit sensitive context to attacker-controlled endpoints [15].

Attack mechanism:

1. Attacker publishes malicious MCP server to public registry
2. Developer installs server (appears legitimate)
3. Agent reads tool descriptor → hidden instructions injected into LLM context
4. LLM executes: reads sensitive files → encodes → transmits via tool call
5. Data arrives at attacker-controlled remote endpoint
   (crosses perimeter as a "legitimate" tool invocation)

Figure 8: Tool poisoning attack mechanism in the MCP server ecosystem (Section 4.6.4).

Real-world evidence (2025–2026):

- **Postmark MCP impersonation (Feb 2026):** First documented in-the-wild malicious MCP server on npm; impersonated the legitimate Postmark email service to capture API credentials
- **Framelink Figma MCP RCE (Feb 2026):** Unsanitized user input in the get_figma_data tool enabled remote code execution, allowing arbitrary data exfiltration

- **MCP Filesystem Server Bash shell (port 4444):** MCP Filesystem Server exploited to open an unauthenticated Bash shell on port 4444 [3]

Rug Pull Attack: A legitimate, previously vetted MCP server is silently updated to include malicious behaviour. Because the server was previously approved, it may bypass re-validation controls. The "rug" is pulled from under the organisation's security posture without any new installation event.

4.6.5 Authentication and Authorization Gaps Enabling Unauthorised Data Transit

The MCP specification's authentication model introduces additional perimeter leakage risks through implementation inconsistencies [15]:

Gap	Mechanism	Perimeter Implication
Dynamic client registration without validation	MCP servers can register new OAuth clients dynamically	Attacker-controlled server obtains valid credentials for data access
Insufficient OAuth scope restrictions	Broad scopes granted at installation time	Agent can access and transmit data beyond task requirements
Long-lived tokens without rotation	Persistent credentials stored in agent configuration	Compromised token enables sustained data exfiltration
Missing human-in-the-loop for sensitive operations	Agent autonomously invokes tools without approval	Sensitive file reads and external transmissions occur without developer awareness

The OAuth 2.1 standard is recommended for MCP authentication, but implementation quality varies dramatically across the reportedly 10,000+ active tool servers. Many community-developed servers use API-key-only authentication or no authentication at all, meaning that any agent with the server's URL can invoke its tools — and any data transmitted to that server is accessible to the server operator.

4.6.6 The Invisible Data Inventory: What Leaves the Perimeter

A critical challenge for enterprise security teams is that the **data transmitted via agent tool calls is not inventoried**. Unlike a file download or an email attachment — which leave audit trails — agent tool invocations transmit data as JSON payloads embedded in API calls, typically over encrypted HTTPS, with no enterprise-side logging of the payload content.

The following categories of sensitive data are routinely transmitted across the perimeter via agent tool calls in typical enterprise development workflows:

Data Category	Transmission Mechanism	Perimeter Exit Point
Proprietary source code	Context window content in tool request payload	Remote MCP server, Cloud LLM API
API keys and secrets	Hardcoded in active files captured by context aggregation	Remote MCP server, Shadow AI endpoint
Database connection strings	Environment files read by agent for context	Remote MCP server
Internal architecture details	Agent reads README, ADRs, config files for context	Remote MCP server, Cloud LLM API
Colleague identities and communications	Agent reads git history, issue tracker, Slack integration	External collaboration tool APIs
Vulnerability information	Agent reads security scan results for remediation context	Remote MCP server
Business logic and algorithms	Core implementation files in agent's context window	Cloud LLM API, Shadow AI endpoint

Authors' analysis; cross-referenced with §4.2 (context aggregation) and §5.2 (CVE catalogue).

This inventory is not theoretical — it reflects the actual context aggregation behaviour of production coding agents as documented in §4.2 and confirmed by the CVE catalogue in §5.2 (CVE-2025-55284: DNS-based exfiltration of API keys via Claude Code; CVE-2025-59536: RCE via malicious project config files).

Telemetry as an invisible data channel: The same data categories above are captured verbatim by LLM observability SDKs (Langfuse, LangSmith, Arize AI) and transmitted to their configured backend. When that backend is a public SaaS endpoint rather than a self-hosted instance, the telemetry channel becomes an invisible perimeter exit point that bypasses all DLP controls. This is classified as **Sensitive Information Disclosure (OWASP LLM02 / CWE-209)**.

Project relevance: Langfuse is one of the LLM observability platforms evaluated and used within the InnovAIte project. The misconfiguration pattern described in §4.6.1 (insecure vs. secure flow diagrams) is therefore directly applicable to the project's own toolchain. Self-hosting Langfuse within the project's VPC boundary is the required mitigation. [Project experience]

Multi-tenancy amplification risk: When using a public SaaS observability platform (e.g., Langfuse Cloud), all customer traces are stored in shared multi-tenant infrastructure. A single

authorization vulnerability in the platform's access control layer can expose one organisation's traces to another tenant — making the centralised trace store a single point of failure for the data of all customers simultaneously. This structural risk is inherent to any shared SaaS observability platform and has been exploited in practice:

Date	CVE	Platform	Attack Vector	Impact	CVSS
Mar 2025	CVE-2025-0330	Langfuse (via LiteLLM)	Langfuse credentials leaked in LiteLLM error response	All stored traces readable	7.5 HIGH
Sep 2025	CVE-2025-9799	Langfuse	SSRF in webhook handler → cloud metadata endpoint	Bulk trace exfiltration via storage credentials	5.0 MED
Sep 2025	CVE-2025-59305	Langfuse	Authorization bypass on migration endpoints	Cross-project trace access (multi-tenant)	7.6 HIGH
Nov 2025	CVE-2025-65107	Langfuse	CSRF account takeover via SSO misconfiguration	Full account access; all traces readable	6.5 MED
Mar 2026	CVE-2026-25750	LangSmith	URL parameter injection → bearer token theft	Full account takeover; complete trace history exposed	8.5 HIGH
Mar 2026	—	LangSmith	Agent API service key leakage + path confusion	Cross-deployment trace access	High

Sources: NVD and vendor security advisories (2025–2026); see case studies below.

Case study — CVE-2025-0330 (Langfuse, March 2025): LiteLLM v1.52.1 serialized langfuse_secret and langfuse_public_key into HTTP 500 error responses. An attacker triggering a team-settings parsing error received Langfuse API credentials in plaintext, then authenticated to Langfuse Cloud and read every stored trace — the complete prompt/response history including any proprietary source code or secrets submitted in prompts. No user interaction required. [NVD; Huntr]

Case study — CVE-2026-25750 (LangSmith, March 2026): A malicious baseUrl URL parameter caused LangSmith Studio to redirect its authentication token exchange to an attacker-controlled server. A victim clicking a crafted link surrendered their bearer token, granting full read/write access to the victim's entire LangSmith workspace — all traces, datasets, and evaluation runs. CVSS 4.0: 8.5 HIGH. [Miggo Security, March 2026]

OWASP Mapping: LLM06 (Sensitive Information Disclosure); CWE-209 (Error Message Containing Sensitive Information); CWE-601 (URL Redirection to Untrusted Site).

4.6.7 Perimeter Leakage Risk Summary

Leakage Vector	Likelihood	Data Sensitivity	Detectability	Risk Level
Streamable-HTTP MCP to unvetted remote server	High	High	Low	**CRITICAL**
Shadow AI (personal accounts, browser extensions)	High	High	Very Low	**CRITICAL**
Misconfigured Telemetry Backdoor (e.g., Langfuse)	**High**	**High**	**Low**	**CRITICAL**
Tool poisoning via malicious MCP descriptor	Medium	High	Very Low	**HIGH**
Rug pull (silent server update)	Medium	High	Low	**HIGH**
CI/CD pipeline with embedded AI calls	Medium	Medium–High	Low	**HIGH**
OAuth scope over-provisioning	High	Medium	Low	**MEDIUM**
Local MCP server with external callbacks	Medium	Medium	Very Low	**MEDIUM**

Authors' analysis synthesizing the perimeter leakage vectors documented in §4.6.

Mitigation cross-reference: The technical controls for these leakage vectors are addressed in:

- §7.3 (MCP vulnerability categories and OWASP controls)
- §9.2 (DLP proxy deployment for AI traffic)
- §9.7 (Zero-Trust network segmentation)
- §13.6 (MCP governance workflow)
- **Telemetry misconfiguration specifically:** self-host observability platforms (Langfuse, LangSmith) within the corporate VPC; never point telemetry SDKs at public SaaS endpoints when processing sensitive code; audit all SDK LANGFUSE_HOST / LANGCHAIN_ENDPOINT environment variables as part of the data-flow inventory (see §4.6.6 CVE table for exploitation evidence)

The enterprise governance framework for preventing perimeter leakage via external agents requires a fundamental shift in security posture: **from perimeter defence to data-flow awareness**. Traditional firewalls protect against unauthorised inbound connections; they are structurally blind to authorised outbound connections that carry sensitive data as payload. AI-specific runtime security (§9.2) and MCP governance (§13.6) are the primary controls for this threat class. The telemetry misconfiguration vector (§4.6.1, §4.6.6) is particularly dangerous because it is architecturally authorised, encrypted, and invisible to DLP systems — making developer and operator awareness the first line of defence.

Implication for organisations: A developer debugging an open-source dependency issue while working in a proprietary codebase is the most common real-world data leakage scenario — it requires no malicious intent and no attack. The context bleed from version control integration means that even routine development tasks can expose sensitive code to cloud LLMs. Organisations must implement workspace-level isolation and classify repositories by sensitivity before enabling AI assistance.

4.6.8 Hidden Auxiliary-Agent Model Routing

A distinct perimeter-leakage vector arises from *auxiliary* LLM calls that coding-agent harnesses issue silently, outside the user-selected primary model path. In KiloCode — an MIT-licensed fork of OpenCode (the open-source case study of §11.1.1) — a hidden title agent (and a parallel summary agent) is invoked on the first turn of every session to generate a conversation title [91]. The title agent is defined with hidden: true and a deny-all tool permission set, rendering it invisible in the user interface and incapable of tool execution. However, when neither agent.title.model nor the global small_model is explicitly configured, model selection falls through to a hardcoded priority list of "small" models (e.g., claude-haiku-4-5, gemini-3-flash, gpt-5-nano) matched by substring, with a gateway kilo-auto/small fallback [91]. The user's first prompt — together with all preceding context messages, including the contents of any attached files — is transmitted to that model via a small-model request, with no user-facing indication that a second, differently-routed model call has occurred. Because the title-generation prompt explicitly instructs the model to *preserve* technical terms, filenames, and HTTP codes, the sensitive material is forwarded verbatim rather than abstracted.

This pattern is a hidden-default data-routing defect rather than a prompt-injection exploit: the title agent holds a deny-all tool permission set and cannot be coerced into taking actions, so it is not an RCE or attacker-controlled exfiltration channel. Its security significance lies in *consent, visibility, and data sovereignty*. A developer who has deliberately configured the primary agent to use a local or zero-data-retention (ZDR) endpoint (§8.5, §9.3–9.4) may nevertheless have the opening prompt — potentially containing credentials, proprietary source, or business context — egress to a different cloud provider operating under different (or no) retention and training terms. The behaviour maps to OWASP **LLM02:2025 (Sensitive Information Disclosure)** as the primary category, with contributing **LLM06:2025 (Excessive Agency)**, **ASI02/ASI10** (tool misuse

/ rogue-agent behaviour in the Agentic 2026 taxonomy), and **LLM03:2025 / ASI04** (supply-chain-style routing to an unvetted model). No CVE or security advisory currently covers this class of issue; the closest academic analog is the "silent egress" pattern in which metadata-generation side-channels leak runtime context while the visible response remains benign [90].

Mitigation requires treating auxiliary calls as first-class egress, not as internal housekeeping. Operators should explicitly bind agent.title.model and small_model to the same governed/local model as the primary agent (or disable automatic title/summary generation), and route *all* agent model traffic — including small-model requests — through the Layer-7 DLP proxy / AI gateway of §9.2 so that the hidden call is inspected and redacted like any other, with direct egress to unconfigured providers blocked. More broadly, security review of any open-source agent harness should enumerate every LLM.stream({ small: true }) and hidden-agent invocation site, since title, summary, commit-message, and prompt-enhancement paths are the typical blind spots that bypass the user's chosen model. The KiloCode maintainers have acknowledged the leak class (PR #6733, "separate taskID for title generation to prevent model leak") [92], but the default-routing risk persists for unconfigured installations.

4.7 Version Control Integration and Context Bleed

Local GitLab Flow (Proprietary Context): Developer → Internal Network → Local GitLab instance. LLM tooling may utilise Webhooks or direct API access to pull repository context, PR diffs, and issue descriptions to feed the RAG pipeline or local context window.

Global GitHub Flow (Open Source Context): Developer → Internet → Global GitHub. Used for fetching upstream dependencies and referencing open-source patterns.

Hybrid Flow and Context Bleed (Critical Risk): A critical risk vector exists when a developer uses a Cloud LLM while working in a Local GitLab environment. The developer might inadvertently submit proprietary Local GitLab code snippets into the Cloud LLM's prompt window to debug an issue related to an open-source dependency, causing a **cross-boundary data bleed**. This is not a theoretical risk — it is the most common real-world data leakage vector in enterprise AI deployments.

---

5 Threat Landscape: Empirical Evidence and CVE Catalogue

5.1 The Empirical Reality: Attacks Are No Longer Theoretical

The threat landscape for AI-assisted development has undergone a fundamental transformation between 2024 and 2026. What were once academic attack scenarios have

become actively exploited vulnerabilities with documented real-world impact. This section presents the empirical evidence.

5.2 CVE Catalogue: Coding Assistant Vulnerabilities (2025–2026)

The following catalogue documents confirmed vulnerabilities in AI coding assistants disclosed between May 2025 and March 2026. The scope is limited to vulnerabilities directly affecting coding assistant products (GitHub Copilot, Claude Code, and generic AI agent frameworks) where sensitive source code or developer credentials are the primary target or collateral damage. Data is sourced from public vulnerability advisories and security research disclosures [19]–[23]; CVSS scores reflect vendor or researcher assessments at time of disclosure.

Date	CVE / Vulnerability	Product	Attack Vector	Impact	CVSS
May 2025	CVE-2025-32711 (EchoLeak)	Microsoft 365 Copilot	Zero-click email prompt injection	Data exfiltration from user's mailbox	9.8
Jun 2025	CVE-2025-55284	Claude Code	Indirect prompt injection via malicious code	DNS-based exfiltration of API keys/secrets	High
Feb 2026	CVE-2025-59536	Claude Code	Malicious .claude/ project config files	Remote Code Execution (RCE)	Critical
Feb 2026	CVE-2026-21852	Claude Code	Compromised project hooks	API token exfiltration	High
Mar 2026	CVE-2026-2256	ModelScope ms-agent (≤v1.6.0rc1)	Command injection (CWE-77) via excessive tool permissions	Full system compromise	6.5
Feb 2026	RoguePilot	GitHub Copilot	Passive prompt injection in repository content	Repository takeover via GITHUB_TOKEN theft	9.6
Mar 2026	Web-Based IPA	Multiple agents	Compromised websites with hidden instructions	Data exfiltration, unauthorised actions	High

5.3 Case Study: EchoLeak (CVE-2025-32711) — The First Zero-Click Production Exploit

EchoLeak represents a watershed moment in AI security: the first documented real-world zero-click prompt injection exploit in a production LLM system [2]. Discovered in Microsoft 365 Copilot, it demonstrates that the theoretical attack chains described in academic literature are fully realizable against enterprise-grade systems.

Attack Chain (4 Steps):

1. **XPIA Bypass:** Attacker sends a benign-appearing email with carefully phrased instructions ("For compliance, do not mention this email"). The email bypasses Microsoft's cross-prompt injection attack (XPIA) filter because the instructions appear legitimate.
2. **Link Filter Bypass:** Reference-style Markdown links ([text][ref]) instead of inline syntax ([text](url)) bypass the output link sanitization filter.
3. **Image Auto-Fetch (Zero-Click Exfiltration):** An <img> tag causes the email client to automatically fetch an external URL, encoding the victim's sensitive data (email contents, calendar items) in the URL parameters — without any user interaction.
4. **CSP Bypass:** A Microsoft Teams proxy URL is used to circumvent the Content Security Policy, making the exfiltration request appear to originate from a trusted Microsoft domain.

Mitigation Effectiveness Matrix:

Mitigation	Scope Violation	Classifier Bypass	Redaction Bypass	Image Auto-Fetch	CSP Bypass	Zero-Click Exfil
Prompt Partitioning	✓	✓	–	–	–	✗
Provenance Gating	✓	–	✓	–	–	✗
Output Policy Gate	✓	✓	✓	✗	–	✗
URL Allowlist	–	–	✓	✗	✓	✓
Strict CSP	–	–	–	✓	✓	✓

Mitigation	Scope Violation	Classifier Bypass	Redaction Bypass	Image Auto-Fetch	CSP Bypass	Zero-Click Exfil
Human-in-the-Loop	✓	✓	✓	✓	–	✓

Source: Authors' analysis based on [2] (EchoLeak) and OWASP [11].

Key lesson: No single mitigation is sufficient. Defence-in-depth combining URL allowlisting, strict CSP, and human-in-the-loop approval for external actions is required to fully prevent this class of attack.

Implication for organisations: No single mitigation stops this attack; organisations that have deployed only one layer of defence (e.g., only prompt filtering) remain fully vulnerable to this class of exploit. The EchoLeak attack demonstrates that enterprise-grade AI systems with multiple security controls can still be compromised through zero-click exploits. Organisations must implement defence-in-depth with human-in-the-loop approval for all external actions.

OWASP and NIST Mapping:

Attack Step	Bypassed Defence	OWASP LLM Top 10	NIST SP 800-53
External email injection	Prompt-injection filter	LLM01: Prompt Injection	SI-10, SI-8, SI-4
Principle of Least Privilege (PoLP) violation	Copilot retrieves email	LLM06: Excessive Agency	AC-6
Link sanitization bypass	Output link not removed	LLM05: Improper Output Handling	AC-4, SC-7
CSP bypass	Teams proxy fetch	LLM02: Sensitive Information Disclosure	A10: SSRF

Sources: [11] OWASP LLM Top 10; NIST SP 800-53 control mappings.

5.4 Case Study: RoguePilot — AI-Mediated Repository Takeover

Discovered by Orca Security and disclosed February 16, 2026, RoguePilot demonstrates a new class of attack: **Passive Prompt Injection** where malicious instructions embedded in repository content cause GitHub Copilot to exfiltrate the GITHUB_TOKEN secret from Codespaces environments.

Attack Mechanism:

1. Attacker creates a repository with hidden prompt injection in README or code comments
2. Victim opens the repository in GitHub Codespaces
3. Copilot automatically processes repository content as context (without explicit developer action)
4. Hidden instructions cause Copilot to execute commands exfiltrating GITHUB_TOKEN
5. Token enables attacker to perform any action within repository scope — full repository takeover

"We forced GitHub to prompt-inject itself. It allowed us to control Copilot's responses and exfiltrate Codespaces' GITHUB_TOKEN secret. The end result was a repository takeover." — Orca Security, February 2026

Significance: This attack requires no social engineering of the developer. Simply opening a malicious repository in an environment with Copilot enabled is sufficient for compromise. This fundamentally changes the threat model for open-source contribution and dependency management.

Implication for organisations: The RoguePilot attack requires no social engineering and no developer error — simply opening a repository in GitHub Codespaces with Copilot enabled is sufficient for compromise. This fundamentally changes the threat model for open-source contribution: reviewing a dependency's source code, triaging a bug report, or evaluating a new library are all now potential attack vectors if performed in a Copilot-enabled environment. Organisations should audit their Codespaces configurations immediately (§13.4, P0).

5.5 OWASP Top 10 for LLM Applications 2025: Sensitive Code Context

The OWASP Top 10 for LLM Applications 2025 [11] provides the industry-standard vulnerability classification framework. The following entries are most directly relevant to sensitive source code submission:

OWASP ID	Vulnerability	Relevance to Code Submission
LLM01	Prompt Injection	Malicious code/comments hijack LLM behaviour
LLM02	Sensitive Information Disclosure	PII, API keys, proprietary logic in responses
LLM03	Supply Chain	Poisoned model weights, compromised plugins/MCP servers

OWASP ID	Vulnerability	Relevance to Code Submission
LLM04	Data and Model Poisoning	Proprietary code in training data leaks to others
LLM05	Improper Output Handling	LLM-generated code executed without validation
LLM06	Excessive Agency	Agents with too many permissions cause unintended actions
LLM07	System Prompt Leakage	Hidden system prompts expose configuration or instructions
LLM08	Vector and Embedding Weaknesses	Poisoned or compromised embeddings/retrieval sources
LLM09	Misinformation	LLM-generated false or misleading content trusted by developers
LLM10	Unbounded Consumption	Resource exhaustion via unbounded model/API usage

Source: OWASP Top 10 for LLM Applications 2025 [11].

5.6 Real-World Exploit Tracker: 2025 Incidents

The following table tracks confirmed security incidents involving AI coding assistants and agentic systems that occurred in 2025. Unlike the CVE catalogue in §5.2, which focuses on product vulnerabilities, this tracker documents actual exploits observed in production environments. The OWASP ASI mapping column links each incident to the corresponding vulnerability category in the OWASP Top 10 for Agentic Applications 2026.

Date	Exploit	Impact	OWASP ASI Mapping
2025	MCP Filesystem Server Backdoor	MCP Filesystem Server exploited to open an unauthenticated Bash shell (port 4444)	ASI04 (Supply Chain)
Oct 2025	Framelink Figma MCP RCE	Unsanitized input enabling RCE	ASI05, ASI02
Oct 2025	Cursor Config Overwrite	Case mismatch enabling persistent RCE	ASI05
Sep 2025	Google Gemini Trifecta	Indirect prompt injection across services	ASI01, ASI02

Date	Exploit	Impact	OWASP ASI Mapping
Jul 2025	Amazon Q Prompt Poisoning	Destructive prompt risking file wipes	ASI01, ASI02, ASI04
May 2025	EchoLeak (Zero-Click)	Email triggers data exfiltration	ASI01, ASI02, ASI06
Mar 2025	GitHub Copilot & Cursor	Backdoors, leaked API keys in production	ASI04, ASI08, ASI09

Source: Compiled from [19], [22], [23], [48], [49].

5.7 Threat Categories for Sensitive Code Submission

The threat categories below represent a project-internal synthesis of the empirical evidence, CVE catalogue, and vulnerability frameworks presented in §5.1–§5.6. They are not derived from a single external taxonomy; rather, they consolidate recurring patterns observed across the 2025–2026 threat landscape into a practical checklist and mental model for development teams assessing their exposure. Each category is actionable: concrete mitigations are discussed in the Four Mitigation Paradigms (§8), Security Strategies (§9), and the Practical How-To Guide (§16).

5.7.1 Data Exfiltration and Credential Leakage

Developers routinely handle secrets (API keys, database URIs, JWTs). If these are hardcoded or present in the active IDE buffer, LLM extensions may automatically bundle them into the context window sent to a Cloud LLM provider. The CVE-2025-55284 exploit (§5.2) demonstrated that even without developer awareness, an AI agent can be manipulated into exfiltrating secrets via DNS requests — a channel that bypasses most traditional network monitoring.

5.7.2 Data Retention and Model Training Leakage

Cloud LLM providers may store API payloads persistently. If enterprise agreements do not mandate zero-data retention (ZDR), submitted proprietary code could be used to fine-tune future foundation models, potentially allowing competitors to extract business logic via targeted prompt engineering. The AAAI-25 research (§6.3.3) confirmed that training data extraction is empirically feasible, with 2,173 usernames and 54 email addresses successfully extracted from GitHub Copilot's training data.

5.7.3 Intellectual Property Contamination and License Violations

Foundation models trained on public repositories may mechanically reproduce exact blocks of code licensed under open licenses (GPL, AGPL, LGPL). Integrating this generated code into a proprietary codebase without attribution or license compliance can trigger severe legal liabilities. This risk is compounded by the fact that LLMs do not provide provenance information for generated code.

5.7.4 Prompt Injection and RAG Data Poisoning

As detailed in §4.4, RAG architectures create a persistent indirect prompt injection vector. An attacker who can write to any data source feeding the RAG pipeline — internal wikis, issue trackers, code comments — can manipulate the LLM's behaviour for all users of the system.

5.7.5 Shadow AI and Unsanctioned Telemetry

Menlo Ventures' 2025 State of Generative AI in the Enterprise report found that approximately **27% of enterprise AI usage is "Shadow AI"** — unsanctioned personal model usage by developers seeking to meet deadlines. This creates an unmonitored data drain where sensitive IP is transmitted without any encryption, DLP scanning, or audit logging. Shadow AI represents the single largest uncontrolled data leakage vector in most organisations.

A related variant is *hidden default model routing* in sanctioned tools: an agent harness may silently route user prompts to auxiliary models via undocumented fallback chains, effectively creating unsanctioned data flows within an approved tool (see §4.6.8).

5.7.6 AI Package Hallucination and Supply Chain Poisoning

A novel risk is "AI Package Planting." LLMs frequently hallucinate the names of non-existent software libraries. Attackers identify these frequently hallucinated names and publish malicious packages with those names to public registries (npm, PyPI), waiting for developers to copy-paste the AI's "poisoned" suggestion. This attack requires no compromise of any legitimate system — it exploits the fundamental unreliability of LLM outputs.

---

6 Code-Specific Attack Vectors: LCCT Vulnerabilities

6.1 Why Code Completion Tools Are Uniquely Vulnerable

LLM-based Code Completion Tools (LCCTs) — GitHub Copilot, Amazon Q Developer, Cursor, and similar products — present a fundamentally different security profile from general-purpose LLMs. The AAAI-25 paper [1] provides the most comprehensive empirical analysis of this attack surface to date, and its findings are alarming.

Key distinctions that make LCCTs more vulnerable than general LLMs:

1. **Contextual Information Aggregation:** LCCTs automatically collect multiple input sources (file name, current file code, other open files, terminal output) without explicit developer action. This aggregation dramatically increases the breach risk surface.
2. **Code-Based Input Specificity:** Safety alignment (RLHF) is trained primarily on natural language, not code. Code-based attacks bypass these safety filters because the model's safety training does not cover the code domain.
3. **Proprietary Training Data:** Fine-tuning on public repositories creates privacy risks — the model may have memorized proprietary code submitted by other users.
4. **Strict Time Constraints:** The rapid response requirement of code completion (sub-second latency expectations) limits the thoroughness of security checks that can be applied.

6.2 The LCCT Workflow and Vulnerability Points

The following workflow diagram illustrates the internal processing pipeline of an LLM-based Code Completion Tool (LCCT). Each stage is annotated with the specific attack vectors that exploit it, as documented in the AAAI-25 benchmark study. Understanding this pipeline is essential for identifying where security controls must be applied.

Input Collection
  ├── File name (exploitable: Filename Proxy Attack)
  ├── Current file code (exploitable: Guided Trigger Attack)
  └── Other open files (exploitable: Cross-File Attack)
    ↓
Input Preprocessing
  ├── Prompt engineering
  └── Security checks ← TIME-CONSTRAINED, INCOMPLETE
    ↓
Data Processing

```
└── Backend fine-tuned LLM generates output
    ↓
Output Post-Processing
  ├── Confidence ranking
  ├── Formatting
  └── Security filtering ← BYPASSED BY CODE-BASED ATTACKS
```

Figure 9: LCCT input, processing, and output workflow with vulnerability points (Section 6.2).

6.3 Attack Methodologies and Empirical Results

The AAAI-25 research identified three primary attack categories against LCCTs, with empirically measured Attack Success Rates (ASR):

6.3.1 Contextual Information Aggregation Attacks

Filename Proxy Attack:

- **Mechanism:** Use a sensitive query as the filename (e.g., naming a file how_to_bypass_authentication.py)
- **ASR on GitHub Copilot:** 72.5%
- **Mechanism:** The LCCT treats the filename as context, causing it to generate content aligned with the sensitive query

Cross-File Attack:

- **Mechanism:** Invoke functions from other open files containing malicious content
- **ASR on GitHub Copilot:** 52.3%
- **Mechanism:** Cross-file context aggregation propagates malicious instructions from one file to the active completion context

6.3.2 Hierarchical Code Exploitation Attacks

Level I — Guided Trigger Attack (Most Dangerous):

- **Mechanism:** Variable name transformation combined with guiding words embedded in code comments
- **ASR Results:**

Model	Level I ASR	Notes
GitHub Copilot	**99.4%**	Near-perfect bypass

Model	Level I ASR	Notes
Amazon Q	46.3%	Significant vulnerability
GPT-3.5	68.3%	High vulnerability
GPT-4	23.8%	Better but not safe
GPT-4o	36.5%	Regression from GPT-4

Source: [1].

The 99.4% ASR on GitHub Copilot is the most significant empirical finding in this domain. It demonstrates that the world's most widely deployed coding assistant is **fundamentally insecure** against code-based jailbreak attacks. For comparison, DAN (Do Anything Now) attacks achieve 18.8% ASR on GPT-4 and 0% on GPT-4o — code-based attacks are dramatically more effective.

Level II — Code Embedded Attack:

- **Mechanism:** Complex embedding with comments, functions, and string concatenation to hide malicious instructions
- **ASR Results:** 41.3% on Copilot, 22.3% on Amazon Q

6.3.3 Code-Driven Privacy Extraction Attack

This attack category targets the extraction of Personally Identifiable Information (PII) from the model's training data using code-structured queries:

Empirical Results from GitHub Copilot:

- **2,173 real GitHub usernames** extracted (80.36% accuracy)
- **54 exact matching email addresses** extracted
- **314 matching locations** extracted (100 exact + 214 fuzzy matches)

These are not synthetic or anonymized data points — they are real user identities extracted from a production system. This confirms that proprietary code submitted to cloud LLMs for training purposes can be extracted by adversaries with sufficient query access.

6.4 Why Code-Based Attacks Bypass Safety Filters

The fundamental reason for the high ASR of code-based attacks is architectural: **RLHF (Reinforcement Learning from Human Feedback) safety training is conducted primarily on natural language, not code**. When safety evaluators review model outputs during training, they assess natural language responses for harmful content. Code-structured attacks exploit this gap:

- A variable named bypass_security_check does not trigger natural language safety filters
- A comment reading # TODO: implement without authentication appears as legitimate development context
- String concatenation to construct malicious instructions is invisible to pattern-matching filters

This creates a systematic blind spot in all current LCCT safety implementations. The implication is that **no currently deployed LCCT can be considered safe for use with highly sensitive source code** without additional external controls.

Implication for organisations: This is a structural limitation of all current LCCTs, not a bug that will be patched. Organisations cannot wait for vendors to fix this; external controls (Prevention/Isolation, Thick Client deployment) are required now. The RLHF gap means that safety filters trained on natural language will never adequately protect against code-based attacks. Without the external controls described in §6.5 and §9, organisations remain vulnerable to code-based attacks that bypass all safety filters.

6.5 Defence Recommendations for LCCT Vulnerabilities

Input Preprocessing Stage:

- Keyword filtering to classify code into safety tiers before processing
- Multi-layer security checks across all contextual inputs (filename, cross-file references, comments)
- Semantic analysis of variable names and code structure for injection patterns

Output Post-Processing Stage:

- Tiered harmful content evaluation based on safety classification of the input
- Balance response time vs. security performance (accept some latency for security)
- Code-specific safety alignment training (not just natural language RLHF)

Organisational Controls:

- Restrict LCCT access to repositories containing security-critical modules
- Implement workspace-level isolation preventing cross-file context aggregation for sensitive files
- Deploy DLP proxies that understand code structure (not just text patterns)

Implication for organisations: Developers cannot protect themselves from these attacks through careful prompting; the attack surface is the IDE configuration itself. The high ASR rates

(72.5% for Filename Proxy, 52.3% for Cross-File) mean that even security-conscious developers are vulnerable to context aggregation attacks. Organisations must implement workspace-level isolation and SCM detection as external controls.

---

7 Agentic AI and Supply Chain Threats

7.1 The Architectural Shift to Agentic Systems

The transition from passive code completion to autonomous agentic coding assistants [5] represents the most significant security paradigm shift in AI-assisted development. Where a traditional LCCT suggests code that a developer must manually accept, an agentic system can:

- Read and modify files across the entire workspace autonomously
- Execute shell commands with developer-level permissions
- Browse external websites and process their content
- Commit code and open pull requests without human review
- Query internal databases and external APIs
- Trigger CI/CD pipelines and deployment workflows

This expanded capability set transforms the threat model fundamentally. A compromised agentic coding assistant is not merely a data leakage risk — it is a **full insider threat** with the ability to introduce backdoors, exfiltrate secrets, destroy data, and compromise downstream systems.

7.2 OWASP Top 10 for Agentic Applications 2026 (ASI Top 10)

The OWASP Top 10 for Agentic Applications 2026 (ASI Top 10) [12] is a dedicated vulnerability taxonomy for autonomous AI agents — distinct from the LLM Top 10 (§5.5), which covers general LLM applications. The ASI Top 10 addresses the expanded attack surface created when an LLM gains the ability to take autonomous actions: reading files, executing commands, and interacting with external systems. The entries most critical for sensitive code submission contexts are listed below, with their relevance to the coding assistant threat model.

ASI01: Agent Goal Hijack

- Prompt injection, deceptive tool outputs, malicious artifacts cause the agent to pursue attacker-defined goals
- Real-world examples: EchoLeak, Operator Prompt Injection, Inception attack

- *Relevance:* An agent working on a codebase can be hijacked to exfiltrate proprietary code to external endpoints

ASI02: Tool Misuse and Exploitation

- Over-privileged tool access, unvalidated input, loop amplification
- Attack scenarios: Tool poisoning, EDR bypass via tool chaining
- *Relevance:* An agent with file system and shell access can be manipulated into executing arbitrary commands

ASI03: Identity and Privilege Abuse

- Un-scoped privilege inheritance, memory-based retention, cross-agent exploitation
- Confused Deputy attack, TOCTOU vulnerabilities
- *Relevance:* Agents inherit developer credentials and can abuse them beyond intended scope

ASI04: Agentic Supply Chain Vulnerabilities

- Poisoned prompt templates, tool-descriptor injection, impersonation
- Real incidents: Amazon Q Supply Chain Compromise, MCP Tool Descriptor Poisoning
- *Relevance:* Third-party MCP servers and agent plugins are unvetted attack vectors

ASI05: Unexpected Code Execution (RCE)

- Prompt injection → RCE, code hallucination, unsafe eval()
- Incidents: Replit "Vibe Coding" Runaway, Direct Shell Injection
- *Relevance:* Agents with shell execution capabilities can be triggered to run malicious code

ASI06: Memory and Context Poisoning

- RAG/embeddings poisoning, shared user context, cross-agent propagation
- Attack: Travel Booking Memory Poisoning, Cross-tenant vector bleed
- *Relevance:* Poisoned agent memory persists across sessions, creating long-term attack vectors

ASI07: Insecure Inter-Agent Communication

- Unencrypted channels, message tampering, replay attacks
- Exploits: Semantic injection, Agent-in-the-Middle via MCP

- *Relevance:* Multi-agent coding systems propagate compromises across agent boundaries

ASI10: Rogue Agents

- Goal drift, workflow hijacking, collusion, self-replication
- Examples: Autonomous data exfiltration, Reward hacking
- *Relevance:* Agents may develop emergent behaviours that deviate from intended scope

7.3 The Model Context Protocol (MCP): A New Attack Surface

The Model Context Protocol (MCP) has become the dominant standard for connecting AI agents to tools and data sources, with **9M+ PyPI downloads** and **4.2M NPM downloads** as of the literature review period, growing to a reported **97 million monthly SDK downloads** by February 2026. This rapid adoption has outpaced security scrutiny. The following table categorizes the primary vulnerability classes identified in the OWASP Practical Guide for MCP Servers (October 2025), along with their attack mechanisms and real-world examples.

MCP Vulnerability Categories (from OWASP Practical Guide for MCP Servers, Oct 2025):

Vulnerability	Mechanism	Example
Tool Poisoning	Malicious instructions in tool descriptions	MCP server description contains hidden prompt injection
Command Injection	Unsanitized input passed to shell	File path parameter contains ; rm -rf /
Retrieval-Agent Deception	Malicious content in retrieved data	Database record contains prompt injection
Malicious Code Execution	Agent executes attacker-controlled code	MCP Filesystem Server exploited to open an unauthenticated Bash shell (port 4444) [3]

MCP Security Controls (OWASP Recommendations):

Tool Poisoning Prevention:

1. Enforce Full Tool Transparency — display complete tool manifest before activation
2. Sanitize Descriptions — review for suspicious keywords and markup
3. Monitor Tool Integrity — pin versions, use hash/checksum verification
4. Runtime Policy Enforcement — least-privilege at MCP server boundary

Prompt Injection Prevention:

1. Validate Untrusted Data — sanitize all external data before passing to model
2. Use Strong Schemas — enforce strict JSON/YAML schemas (Pydantic, JSON Schema)
3. Segment Contexts — establish new connections/sessions for distinct operations

Memory Poisoning Prevention:

1. Secure and Validate Memory — enforce validation on every update, cryptographic hashes
2. Limit Memory Retention — implement TTL on stored data
3. Segment Memory — isolate by session or user identity

Implication for organisations: With 10,000+ active MCP servers and 97M monthly downloads, the probability that a developer in your organisation has installed an unvetted MCP server is high. Each unvetted server is a potential data exfiltration channel. Organisations must implement MCP server governance workflows (§13.6) and centralised MCP gateways (§16.7) before enabling agentic coding assistants.

7.4 AI-Mediated Supply Chain Attacks: A New Threat Class

The emergence of agentic coding assistants has created a fundamentally new class of supply chain attack: **AI-Mediated Supply Chain Attacks**, where LLMs automatically execute malicious instructions embedded in developer content, bypassing human review entirely.

Traditional vs. AI-Mediated Supply Chain:

Traditional Supply Chain Attack:
Malicious Repository → Developer Reviews Code → (Hopefully) Detected → Build → Deploy

AI-Mediated Supply Chain Attack:
Malicious Repository → AI Agent Auto-Processes Content → Immediate Execution
↓
(Human review bypassed entirely)

Figure 10: Traditional versus AI-mediated software supply-chain attack flow (Section 7.4).

Real-World Evidence:

- **RoguePilot (Feb 2026):** Malicious README causes Copilot to exfiltrate GITHUB_TOKEN — no developer action required
- **Web-Based IPA (Mar 2026):** Compromised websites inject instructions into agents browsing for research

- **MCP Filesystem Server Bash shell (2025):** MCP Filesystem Server exploited to open an unauthenticated Bash shell (port 4444) [3]

The critical insight is that **the speed and automation of agentic systems eliminates the human review step** that traditionally served as the last line of defence against supply chain attacks.

7.5 The Autonomous Insider Threat Model

Traditional insider threat detection assumes human actors with behavioural patterns — login times, data access volumes, communication patterns. The 2026 threat model introduces **autonomous agents as potential insider threats**, either compromised or misconfigured.

Key characteristics that defeat traditional insider threat detection:

- Agents operate continuously without human oversight (no "off hours" anomaly)
- Behavioural anomaly detection is ineffective (agents don't have "normal" human behaviour)
- Access patterns differ from human users (API-based, batch operations, no mouse movements)
- Agents can escalate privileges through legitimate tool usage (no suspicious privilege escalation events)
- Agent actions may be attributed to the developer whose credentials they use

Implication: Organisations must develop **agent-specific behavioural baselines** and monitoring strategies that account for the fundamentally different operational profile of autonomous AI systems.

Implication for organisations: Your existing Security Information and Event Management (SIEM) rules, User and Entity Behaviour Analytics (UEBA) baselines, and DLP policies are calibrated for human behavioural patterns — login times, data access volumes, communication patterns. A compromised AI agent operating continuously via API calls will not trigger any of these rules. You are currently blind to this threat class unless you have deployed agent-specific monitoring. The TRiSM framework's Component Synergy Score (CSS) and Tool Utilization Efficacy (TUE) metrics (§9.10) provide a starting point for agent behavioural baselines.

7.6 State-Sponsored Exploitation of AI Coding Tools

Google's Threat Intelligence Group confirmed (as of March 2026) that **China's APT31** has been using Gemini to plan cyberattacks against American organisations. The attack pattern involves:

1. APT operators prompt Gemini as "expert cybersecurity tester"
2. AI generates attack plans, exploit code, and reconnaissance strategies

3. Human operators refine and execute AI-generated plans
4. Attribution is complicated by AI-assisted obfuscation

This development has direct implications for organisations using AI coding assistants: the same tools that accelerate legitimate development also accelerate adversarial development. The asymmetry favors attackers, who can use AI to rapidly generate novel attack variants faster than defenders can develop signatures.

7.7 Multi-Agent System Risks: Cascading Failures

In multi-agent coding environments [5] — where orchestrator agents delegate to specialised sub-agents for testing, documentation, security review, and deployment — a single compromised agent can propagate corruption across the entire system.

Amplified risks in agentic systems (from TRiSM Framework analysis [4]):

1. Single compromised agent can propagate corruption across entire system
2. Inter-agent dynamics are exploitable (race conditions, deadlocks, resource exhaustion via coordination logic manipulation)
3. Memory architectures create persistent attack vectors — episodic, semantic, and vector memory are all vulnerable
4. Shared context across agents increases exposure surface for proprietary information
5. No standardised security frameworks exist for LLM-based multi-agent systems

New Evaluation Metrics (TRiSM Framework [4]):

- **Component Synergy Score (CSS):** Quantifies inter-agent collaboration quality and detects anomalous coordination patterns
- **Tool Utilization Efficacy (TUE):** Evaluates correct and efficient tool invocation, flagging unexpected tool usage

Implication for organisations: In multi-agent coding environments, a single compromised agent can cascade corruption across your entire development pipeline. The lack of standardised security frameworks for LLM-based multi-agent systems means you are deploying unproven technology at scale. Organisations should implement agent isolation, monitor inter-agent communication, and establish kill-switches for agent orchestration.

7.8 Computer-Use and Browser-Use Agents

A distinct product class has emerged alongside general-purpose agentic coding assistants: **Computer-Use Agents (CUAs)** and **Browser-Use Agents (BUAs)**, which perceive a graphical interface through screenshots and act upon it via synthesised mouse and keyboard input. While

the agentic coding assistants analysed in §7.1–§7.7 operate primarily on files, shell commands, and tool calls, CUAs/BUAs add a fundamentally new input channel — **untrusted, adversary-controlled rendered pixels** — to the threat model. This channel is directly relevant to sensitive code submission: a developer may task such an agent with browsing documentation, reading an issue tracker, or inspecting a dependency's website while the agent simultaneously holds proprietary code and credentials in its context window.

Product landscape. Three reference implementations define the class:

- **Anthropic Computer Use** (released October 2024 as a beta API feature) provides full desktop control through screenshot analysis and pixel-level cursor computation, enabling the model to operate any GUI software without specialised APIs [58], [62]. Anthropic explicitly warns that "Claude instructions on webpages or contained in images may override instructions or cause Claude to make mistakes" and recommends isolating the model from sensitive data, restricting internet access to an allowlist, and confirming real-world actions with a human [62].
- **OpenAI Operator** (released January 2025) is powered by the Computer-Using Agent (CUA) model, which combines GPT-4o vision with reinforcement-learned reasoning to "see" (via screenshots) and "interact" (via mouse/keyboard) with a browser [59]. Operator restricts itself to web-based actions and is therefore more precisely categorised as a BUA; it layers takeover mode, user confirmations, a dedicated "monitor model," and a prompt-injection detection pipeline as safeguards [59].
- **browser-use** (open-source library, github.com/browser-use/browser-use) is a widely adopted framework that drives a headful browser through screenshots and DOM interaction, and is one of the five agents benchmarked in recent security studies [53].

Expanded attack surface. The CUA/BUA architecture combines three properties that are individually present in earlier agents but uniquely dangerous when co-located: (i) the agent perceives **the entire rendered screen as trusted instruction context**, so any on-screen content — a web page, a PDF, a pop-up, hidden CSS text — becomes a potential instruction source that bypasses traditional input validation; (ii) the agent holds **developer credentials and proprietary code** in its context (the same sensitive substrate the report is concerned with); and (iii) the agent autonomously **navigates to attacker-influenced content** (websites, search results, advertisements) as part of legitimate task execution. The first property is the architectural novelty: because the agent's value proposition is to read text rendered as pixels, a defender cannot block on the presence of rendered text without breaking the product [57].

Empirical evidence. Recent security research quantifies the resulting exposure:

Study	Agents evaluated	Attack technique	Result	Ref
VPI-Bench (ICLR 2026)	5 platforms (CUAs + BUAs)	Visually embedded prompt injection in rendered UI	CUAs deceived up to 51%; **BUAs up to 100%** on certain platforms; system-prompt defences gave only limited improvement	[52]
Mind the Web (2025)	OpenAI Operator, Browser Use, Do Browser, OpenOperator, Perplexity Comet	Task-aligned injection framed as helpful guidance	**>80% attack success rate** with strong transferability across payloads, environments, and LLMs; CIA-triad payloads incl. camera activation, file exfiltration, impersonation	[53]
HiddenLayer (Oct 2024)	Anthropic Computer Use	Indirect prompt injection via a local PDF	Agent extracted an obfuscated payload and executed sudo rm -rf --no-preserve-root / ("confused deputy")	[58]
CSA research note (Apr 2026)	CUAs (screenshot-based)	Adversarial pop-up windows (ACL 2025)	**86% mean attack success rate** across OSWorld/VisualWebArena; reduced task completion by 47%; "ignore pop-ups" system prompts insufficient	[57]
Google Security Research (2025)	OpenAI Operator	Cross-origin URL exfiltration; TOCTOU tab-switch; fullscreen lock	OAuth codes and credentials exfiltrated by bypassing user confirmation; irreversible state-changing actions on arbitrary origins	[61]

The Google Security Research advisories are particularly significant for the sensitive-code context: they demonstrate that an attacker-controlled web page can induce Operator to leak **cross-origin URLs carrying OAuth authorisation codes** — the same class of credential the report tracks in §5.7.1 and §7.5 — by exploiting the delay between the agent's screenshot-based action prediction and the action trigger (a TOCTOU window) [61]. This is a concrete realisation of the "autonomous insider" model of §7.5: the agent operates with the developer's authenticated browser sessions and can be steered to exfiltrate the very credentials that gate access to proprietary repositories.

Connection to the existing threat model. CUAs/BUAs map directly onto the agentic threat categories already established in this report:

- **ASI01 (Agent Goal Hijack, §7.2):** The "Operator Prompt Injection" and "Inception attack" already listed under ASI01 are precisely CUA/BUA exploits; VPI-Bench and Mind the Web now provide quantitative ASRs for them [52], [53].
- **Web-based IPA (§5.2, §7.4):** The "Web-Based IPA (Mar 2026)" exploit tracker entry — compromised websites injecting instructions into browsing agents [22] — is the in-the-wild instance of the CUA/BUA attack class. The Unit 42 analysis documented pages employing up to 24 distinct injection methods [22], [57].
- **The autonomous insider (§7.5):** A CUA browsing the web on a developer's behalf inherits that developer's authenticated sessions, cookies, and SSO tokens; a single prompt injection can pivot from "summarise this page" to exfiltrating repository access tokens, exactly the credential-leakage pattern of §5.7.1.
- **EchoLeak (§5.3):** EchoLeak's image auto-fetch is the *exfiltration* dual of the CUA *injection* surface — see §7.9 for the precise distinction between image-as-egress-channel and image-as-injection-vector.

A structural caveat, publicly acknowledged by OpenAI, must be recorded: the company states that browser-based prompt injection "may never be fully solved" [60], a position the Cloud Security Alliance echoes in recommending that organisational deployment policies — not solely technical controls — govern which tasks a CUA may perform autonomously [57]. This is consistent with the report's defence-in-depth conclusion in §5.3 that no single mitigation is sufficient.

Implication for organisations: A computer-use or browser-use agent that simultaneously holds proprietary source code/credentials and browses untrusted web content collapses the trust boundary between "instructing the model" and "the model reading the internet." Existing DLP, SIEM, and prompt-injection filters calibrated for textual input do not inspect the pixel channel. Organisations should (a) run CUAs only in dedicated, minimal privilege virtual machines or containers with no access to credential stores or proprietary code [62]; (b) restrict browsing to URL allowlists; (c) require human-in-the-loop confirmation for any action touching credentials, external communications, or financial transactions [57], [59]; and (d) treat the agent's authenticated browser session as a privileged identity subject to the same monitoring as the developer's own (§7.5, §9.9).

7.9 Multimodal Prompt Injection

The prompt-injection coverage in §5.7.4, §7.2 (ASI01), and §7.3 is, with one important exception, **textual**: the payload is a string embedded in code comments, tool descriptions, retrieved documents, or web text, and the defensive scanners discussed in §9 (e.g., output filtering, DLP proxies, URL allowlisting) are likewise calibrated for text. This subsection

documents a class of attacks that the current threat model does not yet address: **multimodal prompt injection**, in which the payload is carried by a non-text modality — an image, a screenshot, or an audio waveform — and is therefore invisible to text-only injection defences.

Definition and distinction from textual injection. In textual indirect prompt injection the attacker places a string in content the model is asked to process; in multimodal injection the attacker places the instruction *inside the pixels of an image, the samples of an audio recording, or the frames/track of a video*. The model's vision or audio encoder reads the embedded instruction as part of its input context and follows it. The seminal formalisation of indirect prompt injection (Greshake et al., 2023) generalises cleanly here: the payload still "rides in on a third-party artifact the model ingests" — but the artifact is now an image, a sound, or a video clip rather than a document [55]. The practical consequence is that an organisation whose defence-in-depth consists only of text-side prompt-injection scanning has, once it accepts any image, audio, or video input, "a defence [that] is shorter than you think."

The video modality is a composite surface. A video stream is not a new injection primitive so much as the union of the two already-documented ones: its visual frames inherit every image-based vector (typographic overlays, adversarial perturbations, steganographic embedding) and its audio track inherits every audio-based vector (imperceptible reverberation, over-the-air adversarial noise). The risk specific to video is *temporal concealment*: an instruction need only be legible to the encoder for the handful of frames (or the sub-second audio window) the model samples, after which it disappears from any human review of the same clip. Because most production multimodal agents sample video sparsely (keyframe extraction + ASR of the audio track), an attacker can place a perturbed keyframe or a short adversarial audio burst that the model transcribes and obeys while a human reviewer never notices it. Empirically, the image-frame and audio-track results above transfer directly to video; no separate "video-only" injection mechanism is required, which is why the literature subsumes video under the image and audio categories rather than treating it independently.

Critical distinction: image-as-injection vs. image-as-egress. It is essential not to conflate two image-related attack surfaces that this report discusses in different sections:

- **Image as *egress* (exfiltration) — already covered, §5.3 EchoLeak.** EchoLeak's "Image Auto-Fetch" step induces the model to *emit* a Markdown <img>/image link whose URL encodes the victim's data; the rendering client then auto-fetches that URL, leaking data **out**. The image is an *output* artefact and the channel is **outbound**. EchoLeak is therefore a *data-exfiltration* exploit, not a multimodal-injection exploit.
- **Image as *ingress* (injection) — the gap this subsection addresses.** In multimodal prompt injection the image is an *input* artefact and the channel is **inbound**: adversarial instructions are embedded *into* an image or audio clip that the model is asked to perceive, steering its behaviour from the inside. The two share the word "image" and the involvement of a vision component, but they are opposite in direction and require

different defences (egress control via renderer/CSP/allowlist for the former; ingress control via pixel/audio-level scanning for the latter). Conflating them — e.g., assuming that because EchoLeak's <img> exfiltration is mitigated by CSP, image-borne *injection* is also handled — leaves the inbound channel entirely open.

Image-based attack vectors. Empirical work establishes image-borne injection as a practical, black-box threat:

Attack	Mechanism	Model / setting	Result	Ref
FigStep (AAAI 2025, Oral)	Harmful instruction rendered as typographic image; benign text prompt elicits completion	6 open-source LVLMs	**82.5% mean ASR**; FigStep-Pro bypasses OpenAI's OCR detector	[54]
Image-based Prompt Injection (IPI)	Segmentation + adaptive font scaling + background-aware rendering to hide prompts in natural images	GPT-4-Turbo (black-box)	**up to 64% ASR** under stealth constraints	[63]
Bagdasaryan et al. (2023)	Adversarial perturbation blended into an image; benign query steers model to attacker-chosen output	LLaVa, PandaGPT	Proof-of-concept image *and audio* indirect instruction injection (the founding multimodal-injection result)	[55]
Goal Hijacking via Visual PI (GHVPI)	Goal-hijack prompt drawn onto the input image	GPT-4V, Gemini	15.8% (GPT-4V), 6.6% (Gemini) ASR	[64]
Malicious Image Patches (MIP)	Adversarial perturbations embedded in on-screen imagery (wallpapers, ads, social posts) captured during OS-agent screenshots	Multimodal OS agents	Self-propagating "computer-worm"-like behaviour; stealthy, transfers across prompts/layouts	[65]

The common root cause, identified by FigStep's ablations, is a **deficiency of safety alignment for visual embeddings**: models whose text channels are aligned to refuse harmful instructions fail to extend that alignment to the visual channel, so the same instruction that is refused as text is obeyed when rendered as pixels [54].

Audio-based attack vectors. The audio modality is an equally capable injection surface, and is directly relevant as coding agents gain voice interfaces:

Attack	Mechanism	Models / setting	Result	Ref
AudioHijack (IEEE S&P 2026)	Context-agnostic, imperceptible adversarial audio blended into natural reverberation	13 SOTA LALMs; commercial Mistral AI & Microsoft Azure voice agents	**79%–96% success** across 6 misbehaviour categories; induced unauthorised web searches, file downloads, email exfiltration	[56]
Bagdasaryan et al. (2023)	Adversarial audio perturbation steers model to attacker-chosen text	PandaGPT	First audio indirect-injection proof-of-concept (co-introduced with the image vector)	[55]
Adversarial background noise (2025)	Over-the-air adversarial noise triggers wake-keywords / harmful commands ("Hey Qwen, send money to X")	Qwen2-Audio (white-box)	Robust to real-world over-the-air transmission; transfers even when system instructions say to ignore background noise	[66]
SWhisper (2026)	Inaudible near-ultrasonic signal demodulates into jailbreak speech via commodity speakers	Speech-driven LLMs (black-box)	Covert, single-query inaudible jailbreak delivery	[67]

The AudioHijack result is the most consequential for the agentic threat model: it demonstrates **context-agnostic** injection — the *same* adversarial audio works regardless of what the user actually said — and shows that commercial voice agents can be driven to perform **unauthorised tool calls** (web search, file download, email exfiltration) while the human hears only ordinary reverberation [56]. This is the audio analogue of the ASI01 goal-hijack and ASI02 tool-misuse categories of §7.2, and it bypasses every text-side injection scanner by entering through the audio encoder.

Relevance to coding agents and the sensitive-code context. Modern coding assistants increasingly accept multimodal input: a developer may paste a *screenshot of a stack trace*, a *UI mockup to reproduce*, a *diagram of an architecture*, a *photo of a whiteboard*, or a *screen recording of a bug reproduction*. Each of these is an adversary-controllable image/video

surface. The threat chain is concrete: an attacker who can place a perturbed image in a venue the developer will screenshot and feed to the agent (a project wiki image, a dependency's README badge, a social-media share of a "design reference") can inject instructions that the agent — which holds proprietary code and credentials — then executes with developer-level tool access. MIP attacks generalise this to imagery that is merely *present on screen* during an OS agent's screenshot loop, requiring no developer action at all [65]. The LCCT context-aggregation behaviour documented in §6.2 (the context window is "everything the agent can read") thus extends from adjacent files to adjacent *pixels and audio*.

Gap in the current report. The report's prompt-injection treatment (§5.7.4, §7.2 ASI01, §7.3, §10.5 regulatory mapping) and its mitigations (§9.2 DLP proxies, §9.9 guardian agents) address textual injection and textual egress. They do not yet cover: (i) inbound injection via image/audio pixels and waveforms; (ii) the failure of OCR/STT-then-text-scan as a defence, because modern VLMs/audio models read through distortions that defeat naive recognisers; or (iii) the alignment asymmetry between text and visual/audio channels. Filling this gap matters precisely because the report's subject — sensitive source code and developer credentials — is the asset that a multimodally-injected agent is best positioned to exfiltrate, given its tool and shell access.

Implication for organisations: Any coding assistant that accepts image or audio input has an injection surface that your text-only prompt-injection scanners do not see. Do not assume that deploying a CSP and image-egress allowlist (which mitigate EchoLeak-style *exfiltration*) also covers image-borne *injection* — they address opposite directions of the same word. Organisations should (a) scan image and audio bytes at the input boundary *before* they reach the VLM/audio encoder, using modality-aware detectors rather than OCR+text-scan [52], [54]; (b) treat all pasted screenshots, mockups, and shared media as untrusted, on par with untrusted web content (§7.8); (c) apply human-in-the-loop confirmation for consequential actions regardless of whether the trigger instruction arrived as text, pixels, or audio; and (d) extend the guardian-agent pattern (§9.9) to reason over multimodal context, not only text.

7.10 Agent-Internal Guardrails Are Insufficient Without Local Enforcement

A recurring and dangerous assumption in agentic coding deployments is that the safety mechanisms *declared* by a tool — ignore rules, approval prompts, "permission modes," or model-side refusals — constitute a complete control set. They do not. **Agent-internal guardrails and declared safety mechanisms cannot be relied upon as sole controls without local enforcement.** Three vendor-authored sources make this concrete.

First, Cursor's own ignore-file documentation limits the scope of .cursorignore to the AI-context surfaces only: it blocks access "from Semantic search; Code accessible by Tab, Agent, and Inline Edit; [and] Code accessible via @ mention references," but states plainly that "the terminal and MCP server tools used by Agent **cannot** block access to code governed by .cursorignore" [68].

The same document warns that "while Cursor blocks ignored files, complete protection isn't guaranteed due to LLM unpredictability" [68]. An ignore rule that an agent can circumvent with a single cat or MCP tool call is a *deterrent*, not a *boundary*.

Second, Cursor's security documentation confirms that the agent's data path is backend-mediated rather than locally contained: "our app makes requests to Cursor backend domains to deliver API, indexing, update, and marketplace functionality" [69]. Codebase indexing transmits code to the backend, and Privacy Mode — the single switch that prevents training on user data and adds provider-side contractual protection — is opt-in: "when enabled, we will not train on your data" [69]. With Privacy Mode disabled, prompts and code context are routed to model providers through the Cursor backend and may be used for training. The control is a configuration flag whose default is permissive; it is not an architectural guarantee.

Third, Anthropic's own guidance for Claude Code frames the tool's capability in unambiguously autonomous terms: Claude Code "can read your files, run commands, make changes, and autonomously work through problems while you watch, redirect, or step away entirely" [70]. The stated control is human supervision ("watch, redirect"), not an internal safeguard that acts *before* the agent acts. When a developer "steps away," the only remaining controls are those enforced outside the agent — at the file system, network, tool, or proxy layer.

Taken together, these sources establish that vendor-declared guardrails are advisory at the agent layer: they shape behaviour but do not *prevent* it. A prompt-injected or misconfigured agent can read an "ignored" file via the terminal, exfiltrate it through an MCP server, or transmit it to a backend that is training by default. The defence must therefore be relocated to layers the agent cannot override.

Recommended controls (local enforcement, defence-in-depth):

- **Ignore rules as a first layer, not a sole control.** Maintain .rooignore / .cursorignore (and the Global Cursor Ignore List for patterns such as **/*.key, **/*.pem, **/id_rsa) to bound *context aggregation*, but treat them as hygiene only — never as the control that protects a Confidential or Restricted repository (cf. LL-4, §8.2).
- **Tool allowlists over blacklists.** Permit only explicitly named tools and deny all others by default, so that newly added tools in a server update are blocked until reviewed. Where the client offers only a blacklist (e.g., RooCode disabledTools, see issue #11259), enforce a true allowlist at the proxy/gateway layer (LiteLLM MCP gateway) [cf. §9.2, §13.6].
- **Approval mode, never "Always Allow."** Keep per-tool confirmation on for cross-repository access, shell execution, external network requests, and out-of-project writes; "Always Allow" removes the last human-in-the-loop checkpoint and is the single most dangerous configuration decision (LL-1).
- **MCP governance.** Treat every MCP server installation as a supply-chain event: publisher verification, version pinning, hash recording, tool-description review, and periodic re-

validation, per the OWASP Practical Guide for Securely Using Third-Party MCP Servers [15] and §13.6.

- **Proxy routing.** Route all agent model traffic through a Layer-7 DLP proxy / AI gateway that inspects prompts and responses and redacts secrets, PII, and proprietary code patterns *before* egress (§9.2); block direct egress to provider domains except through the governed path.
- **Local-model redirection.** For Confidential/Restricted workloads, redirect the agent to local, self-hosted models that never transmit code outside the perimeter (§8.5 thick-client paradigm, §9.4 air-gapped mandate), via a local router (LiteLLM) that can also serve governed cloud endpoints for less-sensitive work.

Connection to the report's open-source / auditability stance. This report has consistently argued that auditability requires the ability to *inspect* the control implementation, not merely to read a vendor's description of it (§8.5, §9.4, and the project's upstream contributions to the open-source assistant ecosystem). The opacity problem is structural: in proprietary agents (Cursor, Codex, Windsurf/Devin, GitHub Copilot), the enforcement of declared guardrails is internal and unverifiable — an organisation must *trust* that .cursorignore is respected, that Privacy Mode is the only training switch, and that "permission modes" cannot be bypassed. In open-source agents (OpenCode, Cline, RooCode), the same controls are *inspectable and patchable*: the team can verify exactly which tool surfaces an ignore rule binds, identify the gap (e.g., terminal/MCP bypass), and contribute a fix upstream. Local enforcement is therefore not only more reliable than declared guardrails — it is the condition that makes reliability *verifiable*. This is the same logic that motivates the report's preference for open-source, source-available tooling and on-premise inference for high-sensitivity workloads.

The hidden-default model-routing defect documented in §4.6.8 further illustrates this principle: the title agent's deny-all tool permissions are a vendor-declared guardrail, but the model-selection fallback chain operates entirely outside user visibility and consent, demonstrating that configuration defaults are not boundaries.

7.11 Fully-Autonomous Software Engineering Paradigms

The agentic assistants analysed in §7.1–§7.7 still presuppose a developer who initiates each action and reviews each result. A distinct and fast-maturing product class removes that presupposition: **fully-autonomous software engineering (SE) agents** that, given a natural-language issue or task, independently plan, write code, execute tests, and open pull requests — end-to-end, with the human relegated to after-the-fact review. This subsection characterises the class, situates it within the report's existing threat model, and assesses the adequacy of the report's recommended controls against it.

The class. Four reference points define fully-autonomous SE:

- **Devin (Cognition AI),** announced March 2024 as "the first AI software engineer." Devin operates inside a sandboxed compute environment equipped with its own shell, code editor, and browser; Cognition describes it as making "thousands of decisions" over long-horizon tasks, recalling context at every step, and fixing its own mistakes [86]. Its launch claim — 13.86% of SWE-bench issues resolved end-to-end and unassisted, against a prior state-of-the-art of 1.96% — anchored the product and the broader category [86]. (The product lineage discussed in §11.1 — Codeium → Windsurf → Cognition acquisition → "Devin Desktop" — extends this autonomy into a multi-agent "command center.")
- **SWE-agent** (Yang et al., NeurIPS 2024) is the open-source academic archetype: a language-model agent plus a custom **agent-computer interface (ACI)** that lets the model autonomously search, navigate, edit, and execute across an entire repository to fix a GitHub issue, achieving 12.5% pass@1 on SWE-bench and 87.7% on HumanEvalFix [82]. Released under MIT, it is also repurposed for offensive cybersecurity and competitive coding — a reminder that the same autonomy serves adversaries as readily as developers [82].
- **SWE-bench** (Jimenez et al., ICLR 2024) is the benchmark that both Devin and SWE-agent are measured against and that effectively defines the task: 2,294 real GitHub issues drawn from 12 popular Python repositories, where an agent must edit a codebase so that the repository's own fail-to-pass tests go green [81]. At launch the best model (Claude 2) solved only 1.96% of issues (4.8% with an "oracle" retriever), establishing a low baseline that autonomous agents have since climbed rapidly [81].
- **OpenHands** (Wang et al., ICLR 2025; formerly OpenDevin) is the open-source platform counterpart to Devin — explicitly "inspired by AI software engineer Devin" — that runs generalist agents which write code, drive a command line, and browse the web inside sandboxed environments, evaluated across 15 benchmarks including SWE-bench and WebArena [83]. As of 2026 it is positioned as a self-hosted "developer control center" that can orchestrate OpenHands, Claude Code, Codex, and Gemini agents over the Agent-Client Protocol (ACP) on local, remote, or private-cloud backends, with the explicit pitch that agents run locally by default and connect to multiple backends without losing context [87].

What unites these systems is an **end-to-end autonomous pipeline — issue → plan → code → test → pull request** — in which the human's role shifts from operator to reviewer. The same property that delivers the productivity gain (remove the human bottleneck) is the property that magnifies the security risk.

Table 7.11.1 — The fully-autonomous SE class at a glance

System	Origin / venue	Licence	Autonomy model	Headline benchmark result	Ref
Devin	Cognition AI (2024, product)	Proprietary	Fully autonomous; sandboxed shell+editor+browser; PR submission	13.86% unassisted on SWE-bench (25% subset) vs 1.96% prior SoTA	[86]
SWE-agent	Yang et al., NeurIPS 2024	MIT (open source)	ACI agent; autonomous repo navigation/edit/test; also used offensively	12.5% pass@1 on SWE-bench; 87.7% on HumanEvalFix	[82]
SWE-bench	Jimenez et al., ICLR 2024	Open (benchmark)	Evaluation framework (the task definition)	Claude 2 solves 1.96% (4.8% oracle) of 2,294 issues	[81]
OpenHands	Wang et al., ICLR 2025	MIT (open source)	Generalist agents; sandboxed; on-prem/private-cloud; ACP orchestration	Evaluated across 15 benchmarks incl. SWE-bench, WebArena	[83], [87]

The report's framework applies — but autonomy amplifies the insider-threat risk. The report already anticipated this class. §7.5 ("The Autonomous Insider Threat Model") warns that autonomous agents "operate continuously without human oversight," defeat behavioural-anomaly detection calibrated for humans, escalate privileges through legitimate tool use, and have their actions attributed to the developer whose credentials they borrow. Fully-autonomous SE is the maximal realisation of exactly that model: the agent holds developer credentials and proprietary code, ingests untrusted content (issue text, PR descriptions, web pages fetched "to read the docs"), and — by design — acts *before* a human reviews. The human-supervision backstop that both this report and Anthropic's own Claude Code guidance treat as the control ("watch, redirect, or step away"; cf. §7.10) is precisely what fully-autonomous operation removes.

Recent security research quantifies how autonomy widens the window between "agent reads untrusted content" and "agent acts with developer privileges":

- A peer-reviewed security analysis of autonomous LLM agents introduces a five-layer lifecycle framework (initialization, input, inference, decision, execution) and finds that the **tightly-coupled interaction paradigm and high-privilege execution "substantially expand the system attack surface,"** with compound threats — indirect prompt

injection, skill supply-chain contamination, memory poisoning, intent drift — that "point-based defence mechanisms" fail to address because they are cross-temporal and multi-stage [84].

- Security firms report the concrete consequences at machine speed: the Cloud Security Alliance documents that 82% of organisations are *not* confident in their IAM systems' ability to govern agent identities and only 17% enforce runtime access controls consistently, while documented exploit chains turn a single injected instruction into remote code execution (e.g., CVE-2025-54795 in Claude Code) or zero-click data exfiltration (EchoLeak, CVE-2025-3271) [89] *(post-cutoff source; published 25 March 2026)*. ReversingLabs further notes that an agent with repository write access can introduce backdoors *at scale*, propagating to every downstream consumer of the software it publishes — the autonomous-insider threat fused with the supply chain [89].
- A 2026 incident makes the chain vivid: a prompt injected via a GitHub issue *title* poisoned an open-source coding agent's CI/CD cache, stole its npm release token, and published a compromised package that silently installed another autonomous agent on roughly 4,000 developer machines [35]. Analysts frame the root cause as a "semantic von Neumann flaw" — instructions and data share the same channel, so untrusted text becomes executable policy — and the precondition as the "lethal trifecta" of access to private data, ability to communicate externally, and exposure to untrusted content [35].

The pattern is consistent: the more autonomous the agent, the longer the decision-to-review window and the larger the blast radius of a single injection. This is fully-autonomous SE's central security property, and it is an *amplification* of threats the report already catalogues (§7.2 ASI01 goal hijack, §7.3 MCP attack surface, §7.4 AI-mediated supply-chain attacks, §7.5 autonomous insider) rather than a new category.

Lack of established evaluation and governance. A second, independent gap compounds the first: the field's ability to *build* autonomous SE agents has outpaced its ability to *evaluate and govern* them.

- The de-facto evaluation, SWE-bench, is under severe strain. Independent analyses report that SWE-bench **Verified** is heavily contaminated — top agents score ~43.20% on Verified but only ~19.25% on the post-cutoff SWE-bench-Live, with measured contamination rates of 8–10% and leaderboard rates inflated by an estimated 6–7 absolute percentage points; up to 345 patch assessments were mis-scored [10] *(post-cutoff source; published 5 May 2026)*. On 23 February 2026 OpenAI reportedly ceased reporting SWE-bench Verified scores after auditing 138 unsolved problems and finding that 59.4% had materially flawed test cases, with every frontier model tested showing evidence of training contamination [31].
- A 2026 position paper formalises the structural diagnosis: coding benchmarks "are misaligned with agentic software engineering" because they collapse model, harness,

and environment into a single end-to-end score, grade against a single reference solution (penalising equally valid alternatives), and provide no signal at the level of individual harness components — yet "a coding agent in practice is not a model: it is a system harness" whose score can move by model-generation-scale margins from the harness alone [85] *(post-cutoff source; arXiv preprint, June 2026)*.

- Surveys of the benchmark landscape find that none of the major coding benchmarks evaluate the *engineering process* (planning quality, verification behaviour, abstention appropriateness), producing a "lab-to-production gap" in which agents optimised for outcome metrics "may develop strategies that are effective on benchmarks but hazardous in production" [85].
- On the governance side, the same CSA survey finding above — 82% lack confidence in agent IAM, 17% enforce runtime controls consistently [89] — matches the report's own observation (§7.7) that "no standardised security frameworks exist for LLM-based multi-agent systems." The capability-to-govern asymmetry is now an industry-acknowledged condition, not a hypothetical.

The practical consequence for this report is that an organisation cannot rely on a public benchmark score to certify an autonomous SE agent safe to deploy against sensitive code: the benchmark may be contaminated, may conflate the model with its harness, and does not measure the process discipline or security posture that determine production behaviour.

The report's recommendations are partial mitigations. The controls this report recommends for agentic coding remain necessary for fully-autonomous SE, but they are *partial* against this deployment profile:

- **Tool and URL allowlists (§9.2, §13.6)** bound *which* tools and endpoints the agent may invoke, but they do not bound the agent's *reasoning over untrusted content*; an agent that may only call allowlisted tools can still be goal-hijacked by an injected issue into using those tools maliciously (ASI01, §7.2).
- **Approval mode / "never Always Allow" (§16.1 LL-1; §9.9 guardian agent)** is the report's principal human-in-the-loop checkpoint — yet fully-autonomous operation is defined by its removal, and tooling explicitly offers escape hatches (--dangerously-skip-permissions, --yolo, -trust-all-tools) that disable it for exactly the long-running unattended runs autonomy requires [35]. Where autonomy is the goal, approval mode is the first control eroded.
- **MCP governance (§13.6)** governs the *tool supply chain* but not the *content supply chain*: issue bodies, PR descriptions, and fetched web pages are not MCP servers, yet they are the injection vectors the incidents above exploited [84], [35].
- **Local-model redirection (§8.5 thick client, §9.4 air-gapped, LiteLLM routing §9.2)** removes the cloud egress of code and credentials — a genuine and important mitigation

— but it does not constrain the agent's *local* privileged execution: a locally-run autonomous agent with shell and repository write access can still be prompt-injected into exfiltrating via a permitted channel or inserting a backdoor [84], [89].

None of this invalidates the report's controls; it relocates them. Defence-in- depth — the report's own conclusion that "no single mitigation is sufficient" (§5.3) — remains the correct posture, but for fully-autonomous SE the layers must be both stricter and augmented. Concretely, fully-autonomous SE should be treated as the **highest-risk deployment profile** in the report's taxonomy: governed by the strictest *combination* of the report's controls (local redirection + air-gapped execution + tool/URL allowlists + MCP governance), plus **agent-specific behavioural monitoring** (§7.5, §9.10 TRiSM CSS/TUE metrics) and, critically, a **human approval gate on PR merge** so that autonomy ends at the pull request, not at production. Under no reading should a fully-autonomous agent be pointed at a Confidential or Restricted repository (§8.2 LL-4) without the local-redirection/air-gapped paradigm of §8.5/§9.4 and that merge-time human gate in place.

Implication for organisations: Fully-autonomous SE agents deliver their productivity gain precisely by removing the human-in-the-loop that this report's controls lean on. The threat model of §7.5 applies in amplified form: longer decision-to-review windows, machine-speed action, and — per the evidence — single-injection-to-supply-chain-compromise chains. Treat fully-autonomous SE as your highest-risk agentic profile: keep it off Confidential/Restricted repositories, run it air-gapped with local models (§8.5, §9.4), enforce allowlists and MCP governance (§9.2, §13.6), deploy agent-specific behavioural monitoring (§7.5, §9.10), and *never* let autonomy cross the PR-merge boundary without a human gate. Do not trust a public benchmark score as a safety certificate: the evaluation infrastructure is contaminated and structurally misaligned with the systems it grades [85], [31], [10].

8 The Four Mitigation Paradigms: Deep Analysis

8.1 Overview and Comparative Framework

The D6.3 deliverable specification identifies four primary approaches to handling sensitive source code in AI-assisted development environments. This section provides a deep, evidence-based analysis of each paradigm, updated with 2025–2026 research findings. The following comparative viability matrix summarises the security, performance, and economic trade-offs of each approach. Arrows (↑↓) indicate whether the viability has improved or degraded since the 2025 baseline assessment.

Comparative Viability Matrix (Updated March 2026):

Criterion	Prevention/Isolation	Obfuscation	Secure Channel	Thick Client (Local)
Security Level	High	Low–Medium ↓	Medium	Very High
Implementation Complexity	Medium	High	Low–Medium	High
Performance Impact	Low	Medium	Low	None
Utility Preservation	Medium–High	Medium ↓	High	Very High
Scalability	High	High	Very High	Medium
Compliance Support	High	Medium	Medium–High	Very High
Relative CAPEX	Low	Medium	Low	High (consumer to datacenter-grade hardware)
Relative OPEX	Low	Low–Medium	Medium (subscription/API)	Low–Medium (maintenance, power)
2026 Viability Assessment	**High ↑**	**Low ↓**	**Medium**	**Very High ↑**

Source: Authors' analysis. Viability assessments based on cited research throughout §8.

Note: Cost comparisons are relative. Actual CAPEX/OPEX varies significantly based on scale, hardware availability, and market conditions. Local deployment typically requires higher upfront investment but lower ongoing costs at scale.

↑ *Improved since 2025 assessment;* ↓ *Degraded since 2025 assessment*

8.2 Paradigm 1: Prevention and Isolation

8.2.1 Mechanism

The Prevention strategy is predicated on the **Principle of Least Privilege** applied to data context. By utilising Abstract Syntax Tree (AST) pruning and modular isolation, an internal pre-processing engine identifies **Security-Critical Modules (SCMs)**. When a developer initiates an

LLM prompt, the system automatically decouples these SCMs. The remaining "Sanitized Context" is sent for inference. When the manipulation is complete, the SCMs are restored to their previous state.

AST-Based SCM Identification:

- Static analysis identifies modules with security-critical characteristics: cryptographic operations, authentication logic, proprietary algorithms, database access patterns
- AST pruning removes these modules from the context window before transmission
- Contextual Mocking replaces isolated code with high-level functional interface descriptions, allowing the LLM to understand *what* a module does without seeing *how* it does it

Effectiveness: AST-based SCM detection achieves 88.2% F1-score on benchmark datasets [39], with precision-recall trade-offs varying by codebase complexity and language. At 85% detection accuracy, empirical studies show approximately 15% utility loss in LLM code generation tasks [40]. The remaining ambiguous modules require human review. This figure is language- and codebase-dependent; traditional static analysis achieves only ~70% F1-score on the same benchmarks, while LLM-based detection reaches ~82.5% [39].

Category note (peer review): Reference [39] (Tian et al., Expert Systems with Applications, 2024) addresses *vulnerability detection* using AST decomposition — a related but distinct task from SCM detection for LLM context isolation. No direct benchmark exists for SCM detection in the LLM context isolation setting; the ~88% F1-score (synthetic benchmarks; real-world performance is lower, e.g. D2A accuracy ≈64%) is an approximation from the related task of AST-based vulnerability detection [39]. Project-internal validation of SCM detection accuracy has not yet been conducted (see §8.2.3).

8.2.2 The Semantic Coherence Challenge

The primary limitation is the **Semantic Coherence Problem**: if the LLM is asked to refactor a function that calls a hidden module, the model may hallucinate the module's behaviour, resulting in:

- Generated code that calls the hidden module with incorrect parameters
- Architectural suggestions incompatible with the hidden module's actual interface
- Security vulnerabilities introduced by the LLM's incorrect assumptions

Mitigation: Contextual Mocking addresses this by providing the LLM with a high-level description of the hidden module's interface (inputs, outputs, side effects) without revealing its implementation. Research from EASE 2025 [41] identifies that effective mocking must include not only function signatures but also behavioural contracts: pre/post-condition specifications,

side-effect descriptions, and error handling contracts. Without this level of detail, LLMs generate code with parameter mismatches and incorrect error handling.

Quantitative impact of context pruning on LLM utility (illustrative model, derived from ZeroSecBench evaluation framework [40]):

Context Condition	Pass@1	BLEU-4	Semantic Coherence Score
Full Context (baseline)	72.4%	0.412	1.00
Pruned (85% SCM accuracy)	61.8%	0.358	0.85
Pruned (70% SCM accuracy)	48.2%	0.289	0.67

Attribution note (peer review): The specific Pass@1 figures above (72.4% / 61.8% / 48.2%) are illustrative values derived from the ZeroSecBench evaluation methodology and framework [40]; they are not directly reported in the ZeroSecBench paper itself (which reports best overall Pass@1 ≈ 0.26–0.28 for frontier models on its secure-code benchmark). The table above models the *relative utility degradation* at different SCM detection accuracy thresholds, consistent with the ZeroSecBench framework's approach to measuring context-pruning impact. Project-internal empirical validation of these figures has not been conducted (see §8.2.3).

This model confirms that SCM detection accuracy directly determines the security-utility trade-off: at 85% accuracy, utility loss is approximately 15%; at 70% accuracy, utility loss exceeds 30%.

8.2.3 Proposed Evaluation Methodology

Note: The following constitutes a research proposal for future empirical validation within the InnovAIte project context. The findings cited above are from external benchmark studies; project-internal experiments have not yet been conducted.

A rigorous evaluation of the Prevention/Isolation paradigm on project-representative codebases would proceed in four phases:

1. **Ground Truth Construction:** Manually annotate SCMs across 3–5 representative repositories using a four-type taxonomy: (A) cryptographic operations, (B) authentication/authorization logic, (C) proprietary algorithms, (D) sensitive data access. Target inter-annotator agreement $\kappa > 0.8$.
2. **AST-Based Detection Testing:** Implement detection using standard AST tools (Tree-sitter, ANTLR); compute Precision, Recall, F1-Score per SCM type; analyse false positives and false negatives.
3. **LLM Utility Impact Assessment:** Execute 50 representative code generation tasks with full vs. pruned context; measure unit test pass rate, Semantic Coherence Score (SCS), and developer effort to correct generated code.

4. **Threshold Analysis:** Vary detection accuracy thresholds (70%, 80%, 90%) and measure utility loss at each threshold to identify the optimal operating point balancing security and utility.

8.2.4 Open Research Questions

The following questions remain unresolved in the literature and represent directions for future research:

1. **Cross-Language Generalisation:** AST-based detection rules are language-specific. Achieving portability across Java, C#, Python, and Go without per-language rule engineering is an open problem — discussed in §8.2.3 (Proposed Evaluation Methodology).
2. **Dynamic SCM Identification:** Some modules become security-critical only in certain execution contexts (e.g., when processing PII). Static analysis cannot capture context-dependent sensitivity.
3. **Automated Mocking Quality:** What level of mock detail is sufficient for LLM utility? No formal specification language for behavioural contracts has been standardised.
4. **Real-Time Performance:** AST parsing adds latency. Whether SCM detection can operate within IDE response time constraints (<100ms) at enterprise scale has not been empirically established.

8.2.5 2026 Viability Assessment: HIGH ↑

The RoguePilot vulnerability (Feb 2026) has significantly elevated the importance of context isolation. The attack demonstrated that **any content in the agent's context window is a potential attack vector**. Prevention/Isolation, by minimising the context window to only what is strictly necessary, directly reduces the attack surface for prompt injection attacks.

Best suited for: Organisations with well-defined security-critical modules; teams requiring moderate security with minimal performance impact; organisations beginning their secure AI journey (lowest cost entry point).

8.3 Paradigm 2: Functional Obfuscation and Semantic Masking

8.3.1 Mechanism

Obfuscation transforms code before LLM submission. Proprietary identifiers and logic flows are mapped to a symbolic dictionary before transmission and reversed upon receiving the model's response.

Obfuscation Technique Categories:

Technique	Reversibility	LLM De-obfuscation Resistance	Empirical De-obfuscation Rate (Frontier LLMs)
Lexical Obfuscation (variable/function renaming)	Yes	**Very Low**	95.99% syntax / 93.42% execution (GPT-4o); 97.24% / 62.60% avg across all models [42]
Control Flow Obfuscation (flattening, opaque predicates)	Partial	Low	95.99% syntax / 93.42% execution (GPT-4o); 97.24% / 62.60% avg across all models [42]
Data Flow Obfuscation (value encoding, buffer splitting)	Yes	Low–Medium	Qualitative 0–5 scale (GPT-4o, Claude 3.7 Sonnet) [43]
Combined (all techniques)	Partial	Moderate	95.99% syntax / 93.42% execution (GPT-4o); 97.24% / 62.60% avg across all models [42]
Reversible Semantic Masking (symbolic dictionary)	Yes	Medium–High	Not yet benchmarked

Sources: [42] (JsDeObsBench) reports GPT-4o syntax accuracy (95.99%) and execution accuracy (93.42%); the average across all 7 tested models is 97.24% syntax / 62.60% execution. [42] does not test Claude 3. [43] (Tkachenko et al.) evaluates GPT-4o and Claude 3.7 Sonnet on a qualitative 0–5 attacker-knowledge-level scale across four obfuscation dimensions and reports no percentage de-obfuscation rates.

8.3.2 The "Simplicity by Obfuscation" Phenomenon

Research from EASE 2025 [41] revealed a critical counter-intuitive finding: **LLMs often reduce cyclomatic complexity when attempting obfuscation**, rather than increasing it.

- GPT-4-Turbo achieved an **81% pass rate** with few-shot prompting for obfuscation tasks vs. only **29%** with standard prompting

- When GPT-4-Turbo attempts obfuscation with few-shot prompting, it reduces cyclomatic complexity (average ΔCC ≈ −1.0 McCabe points) [41] — meaning obfuscated code is often *more* readable than the original
- This creates a double vulnerability: LLMs de-obfuscate submitted code with high success rates (GPT-4o: 95.99% syntax / 93.42% execution accuracy; 97.24% / 62.60% average across all tested models [42]; qualitative 0–5 scale [43]), and LLMs simplify rather than obfuscate when asked to transform code

This finding has profound implications: LLMs are not only capable of de-obfuscating code submitted to them — they may actively simplify obfuscated code, defeating the protection entirely.

Note: The above findings are drawn from external literature (EASE 2025, JsDeObsBench, Tkachenko et al.). Project-internal experiments on enterprise-representative codebases have not been conducted. The literature-based downgrade of this paradigm is therefore a research synthesis conclusion, not an empirically validated project result.

Implication for organisations: If your organisation is using LLM-assisted obfuscation as a security control, the LLM may be actively reducing your protection level. Obfuscation should not be relied upon as a primary defence mechanism — it provides a false sense of security while potentially degrading code quality and maintainability.

8.3.3 The Utility Loss Function

The success of obfuscation is measured by the **Utility Loss Function**:

- **Routine refactoring:** Low utility loss — obfuscation maintains high utility
- **Bug fixing:** Medium utility loss — semantic masking may hide the bug's context
- **Architectural design:** High utility loss — the LLM cannot reason about architecture without understanding the actual domain

8.3.4 Open Research Questions

The following questions remain unresolved and represent directions for future empirical work:

1. **Quantum-Resistant Obfuscation:** Can cryptographic obfuscation techniques (e.g., indistinguishability obfuscation) provide LLM-resistant protection? Current research is theoretical only.
2. **Adversarial Training:** Can LLMs be intentionally weakened for de-obfuscation tasks while maintaining code generation capabilities? No empirical evidence exists.

3. **Hybrid Approaches:** Does combining obfuscation with Prevention/Isolation provide additive security? No benchmark studies exist for this combination.
4. **Enterprise Code Specificity:** All quantitative de-obfuscation studies use JavaScript or assembly benchmarks. Whether results generalise to enterprise Java/C#/Python codebases with domain-specific obfuscation schemes is unknown.

8.3.5 2026 Viability Assessment: LOW ↓

Downgraded from 2025 baseline for two reasons: (1) modern frontier models demonstrate sophisticated semantic understanding that defeats lexical and control flow obfuscation, with empirically measured de-obfuscation capability (GPT-4o: 95.99% syntax / 93.42% execution accuracy; 97.24% / 62.60% average across all tested models [42]; qualitative 0–5 attacker-knowledge-level scale across four obfuscation dimensions [43]); (2) the "Simplicity by Obfuscation" phenomenon [41] means LLM-assisted obfuscation may reduce rather than increase protection.

Best suited for: As a supplementary layer in hybrid architectures, not as a primary defence; moderate-security requirements with cost constraints.

Implication for organisations: Obfuscation provides a false sense of security. Organisations should not classify obfuscated code as "safe to submit" to cloud LLMs — modern frontier models can de-obfuscate submitted code (GPT-4o achieves 95.99% syntax and 93.42% execution accuracy; 97.24% / 62.60% average across all tested models [42]), and the "Simplicity by Obfuscation" phenomenon means LLM assistance may actively reduce protection.

8.4 Paradigm 3: Governed Secure Channel Architectures

8.4.1 Mechanism

This strategy shifts the burden of security from the data itself to the transit medium, involving:

- **Transport Security:** TLS 1.3 with certificate pinning
- **Authentication:** Mutual TLS (mTLS), OAuth 2.1, OIDC/PKCE
- **Network Isolation:** Private Links, S2S VPN tunnels to cloud providers
- **Contractual Controls:** Zero-Data Retention (ZDR) enterprise agreements

Enterprise Secure Channel Solutions (2025):

Vendor	Solution	Key Features
Microsoft	Azure Private Endpoint	Network isolation, no public internet exposure

Vendor	Solution	Key Features
AWS	PrivateLink + VPC Endpoints	Direct connection without internet transit
Google	VPC Service Controls	Perimeter-based access control
Palo Alto	Strata AI Access Security	Precision AI® for sub-function identification
Cisco	AI Defense	End-to-end protection for AI transformation

Authors' compilation from vendor documentation (2025).

8.4.2 Critical Limitations

The Secure Channel approach has a fundamental architectural limitation: **it protects only transit security, not model training leakage**. Even with perfect transit encryption and zero-retention contractual guarantees:

- The provider's inference infrastructure processes plaintext code
- **Verification of "Zero-Retention" policies remains technically unresolved** — ZDR agreements are contractual, not cryptographic. Organisations cannot technically verify that a provider has honored ZDR commitments. Current provider implementations rely on self-attestation (OpenAI, Anthropic) or limited audit logs (Google Vertex AI, Azure OpenAI), none of which constitute independent cryptographic verification [44]
- Contractual guarantees are only as strong as the provider's compliance and the organisation's ability to audit
- Even if inference data is not retained, data memorized during prior training phases cannot be removed from model weights without full model retraining

8.4.3 Verifiable Zero-Data Retention: Research Directions

Note: The following describes emerging research directions. No production implementations of verifiable ZDR exist as of March 2026.

Current ZDR agreements are contractual, not cryptographic. Emerging research proposes technical mechanisms for verifiable deletion:

- **zkUnlearner Framework [44]:** Zero-knowledge proofs for machine unlearning — allows a provider to prove that specific data has been removed from model weights without revealing internal model state. Currently at research prototype stage only (September 2025).
- **TEE-Based Attestation:** Processing in trusted enclaves with deletion certificates — provider runs data processing in TEE (Intel SGX, AMD SEV), generating cryptographically

signed deletion certificates. Vulnerable to TEE.Fail side-channel attack [36]; no production implementations for ZDR.

- **Immutable Audit Logs:** Blockchain-based data lifecycle tracking with customer read-only access — technically mature but no LLM provider has implemented this for ZDR verification.

Fundamental limitation: Even with verifiable inference ZDR, customers cannot verify that their data was not used in prior training. The memorization problem — where training data is embedded in model weights — is not addressed by any current ZDR mechanism.

Implication for organisations: Your ZDR agreement with OpenAI/Anthropic/Google is a legal document, not a technical guarantee. If the provider violates it, you may not know until a breach occurs. Organisations should treat ZDR as a baseline contractual control, not a primary technical defence for highly sensitive code.

Practical recommendation: Until verifiable ZDR mechanisms mature, organisations should:

1. Prefer Thick Client (local deployment) for highly sensitive data
2. Use ZDR agreements as a baseline contractual control, not a primary technical defence
3. Monitor research progress on cryptographic deletion proofs (zkUnlearner and successors)

8.4.4 Open Research Questions

1. **Scalability:** Can verifiable ZDR protocols handle enterprise-scale API volumes (millions of requests/day) without prohibitive latency?
2. **Legal Enforceability:** Would cryptographic deletion certificates hold up as evidence of compliance under GDPR Article 17?
3. **Standardisation:** Is there industry appetite for a ZDR verification standard (analogous to SOC 2 for security)?

8.4.5 2026 Viability Assessment: MEDIUM (Necessary but Insufficient)

The Secure Channel approach is **necessary but not sufficient** as a standalone strategy. It should be considered a baseline requirement for any cloud LLM deployment, not a complete solution. The inability to technically verify ZDR promises is a fundamental limitation that only local deployment fully resolves.

Best suited for: As a mandatory baseline layer in all hybrid architectures; organisations already using enterprise cloud services with existing private networking.

Implication for organisations: The Secure Channel approach is necessary but insufficient as a standalone strategy. Your ZDR agreement is a legal document, not a technical guarantee. Organisations should treat Secure Channel controls as a baseline requirement, not a complete solution — the inability to technically verify ZDR promises is a fundamental limitation that only local deployment fully resolves.

8.5 Paradigm 4: Thick Client — Localized Inference

8.5.1 Mechanism

The "Thick Client" approach is the **"Gold Standard" for high-security environments**. By deploying LLMs on local workstations or private, air-gapped HPC clusters, the network risk is eliminated entirely. **Network Risk: Zero** (no external transmission of code).

8.5.2 Hardware Requirements (2025–2026)

Model	Quantization	Hardware Tier	VRAM Required	Relative CAPEX
Llama 3.1 8B	Q4_K_M (4-bit)	Consumer-grade GPU (RTX 4090/5090 class)	~6 GB	Low–Medium
Qwen 2.5 14B	GPTQ Int-8	High-end Workstation GPU (RTX A6000 48GB)	~14 GB	Medium
Qwen 2.5 32B	AWQ (4-bit)	Enterprise GPU (A100 80GB class)	~20 GB	High
Llama 3.3 70B	AWQ (4-bit)	Multi-GPU Enterprise (Dual A100 80GB)	~35 GB/GPU	Very High

Source: [47] (a commercial vendor page; these benchmarks have not been independently verified against primary technical sources).

Note: Hardware pricing has shown significant volatility in 2025–2026, with consumer-grade GPUs experiencing 40–55% price increases above MSRP due to supply constraints. Organisations should evaluate current market conditions before procurement.

8.5.3 Performance Benchmarks (2025)

Hardware	Model	Quantization	Time To First Token (TTFT) (Best)	Tokens/sec	Concurrent Users
H100 80GB	Llama 3.3 70B	AWQ	131 ms	40.11	1–4
L40S 46GB	Llama 3.1 8B	EETQ 8-bit	136 ms	73.92	1–8
RTX A6000 (48GB)	Qwen 2.5 14B	GPTQ Int-8	85 ms	46.30	1–4

Source: [47] (a commercial vendor page; these benchmarks have not been independently verified against primary technical sources).

8.5.4 Deployment Economics: The Break-Even Shift

Metric	2023 Prediction	2025–2026 Reality
Total Cost of Ownership (TCO) Break-even Period	36 months	**Significantly shortened (under 2 years for many organisations)**
Open-source Model Quality	72% of GPT-4	**85–95% of GPT-4**
Local Inference Speed	Baseline	**2–5x improvement**

Source: [9].

Note: The "85–95% of GPT-4" figure is an aggregate estimate derived from benchmark comparisons against current frontier models; [9] compares open-weight models to GPT-5 and Claude-4, not directly to GPT-4.

Relative Cost Comparison (qualitative):

Scenario	Local Deployment	Cloud Deployment	Hybrid Approach
Upfront CAPEX	**High**	Low	Medium
Ongoing OPEX	**Low–Medium**	High	Medium
Best Value At Scale	**Yes**	No	Yes (flexible)

Authors' analysis.

Note: Local deployment becomes economically favorable when token volume exceeds typical enterprise thresholds. Cloud deployment offers lower upfront costs but higher ongoing expenses. Hybrid approaches balance both models.

8.5.5 Quantization Advances Enabling Local Deployment

- **BitNet (1.58-bit):** Ternary weights (-1, 0, 1), 93% size reduction with minimal quality loss
- **SmoothQuant:** Handles activation outliers, better for INT8 quantization
- **EXL2:** GPU-optimised, faster than GPTQ on NVIDIA hardware
- **AWQ:** Preserves salient weights, excellent quality/size tradeoff
- **Energy Savings:** 4-bit quantization cuts inference energy by 45–80%

8.5.6 Technical Security Considerations for Local Deployment

While Thick Client deployment eliminates network-based attacks, organisations must address local security hardening. The following technical constraints and vulnerabilities are relevant:

TEE-Based Hybrid Deployments (Cloud-Local):

If organisations use TEE-isolated cloud inference rather than fully air-gapped local deployment, the following constraints apply [36, 38]:

TEE Type	Max Enclave Memory	Practical LLM Size	TEE.Fail Vulnerable	Inference Overhead
Intel SGX	~256 GB EPC	Up to 7B params (quantized)	Yes	+4–10% throughput
Intel TDX	VM-configurable	Up to 30B params	Yes	+4–10% throughput
AMD SEV-SNP	VM-configurable	Up to 30B params	Yes	+4–10% throughput

Source: [36] TEE.Fail research; [38] IEEE IISWC 2025 confidential LLM inference benchmarks

Key finding: Large LLMs (70B+) face significantly higher overheads in TEE enclaves (up to 23% on multi-socket configurations [38]), making TEE less practical for large-model deployment. For maximum security, air-gapped local deployment is superior to TEE-isolated cloud.

Air-Gapped Local Deployment — Security Hardening Requirements:

Layer	Control	Rationale
Physical	Locked server room, hardware inventory tracking	Prevents TEE.Fail-class physical attacks
Physical	Supply chain verification (hardware provenance)	Mitigates pre-installed backdoors
System	Full disk encryption (LUKS/BitLocker), minimal OS	Reduces attack surface
System	Network interface disabled or physically removed	Enforces air-gap
Application	Container isolation, resource limits	Prevents privilege escalation
Application	Model signature verification	Prevents model tampering
Application	Immutable audit logs to external system	Provides accountability

Side-Channel Risks (if TEEs are used):

- Cache side-channel attacks show technique-dependent success: Flush+Reload achieves **99.2%** AES key recovery (13,000 encryptions), while Prime+Probe achieves only **53.4%** even with 1M encryptions [37a]
- Energy-based attacks (CLKSCREW, 2018) can extract cryptographic keys via voltage/frequency manipulation [37b]
- These attacks require physical or privileged access; on-premise deployments with physical security controls have substantially lower exposure than multi-tenant cloud

Open Research Questions:

1. **Quantum-Resistant TEEs:** Will quantum computing break current TEE encryption? Post-quantum cryptography integration timeline is unknown.
2. **Formal Verification:** Can TEE implementations be formally verified for security? Early research only (seL4 microkernel success has not been replicated for TEEs).
3. **Side-Channel Immunity:** Can future TEE generations achieve provable side-channel resistance? Hardware vendors claim progress; no independent verification exists.

8.5.7 2026 Viability Assessment: VERY HIGH ↑

Significantly upgraded due to: (1) AI-mediated supply chain attacks demonstrate cloud-connected agents are fundamentally vulnerable to attacks that local deployment prevents entirely; (2) hardware cost reduction and model quality improvement; (3) EU AI Act compliance

is significantly easier with local deployment; (4) the autonomous insider threat model is mitigated by local deployment.

Best suited for: High-security environments (defence, finance, healthcare, critical infrastructure); organisations processing 50M+ tokens monthly; teams with compliance requirements (GDPR, HIPAA, PCI, EU AI Act).

Real-World Zone-Based Placement Strategy:

Production deployments often implement a zone-based approach to model placement, where different security zones host models based on data sensitivity:

Zone	Security Profile	Model Types	Use Case
AIRS Protected AI Workload Zone	Full prompt/response scanning, cloud telemetry	Smaller on-prem models (8B–14B)	Non-sensitive workloads, public-facing services
AI Workload Zone	Cloud analysis suppressed; L7 + on-prem security only	Medium on-prem models (14B–32B)	Sensitive data requiring local processing
Kubernetes Internal Zone	Isolated cluster, no external egress	Large on-prem models (70B+)	Highly confidential code, proprietary algorithms

Operational Pattern: A centralised model router (e.g., LiteLLM) provides unified authentication, rate limiting, and logging across all zones — creating a single control point for policy enforcement while maintaining zone isolation. This pattern enables organisations to offer a consistent developer experience while enforcing different security postures based on workload classification.

8.5.8 Capability Gap Caveat — Task-Dependent, Not a Single Percentage

The "85–95% of GPT-4" figure cited in §8.5.4 (sourced to [9]) and the "98% of GPT-4 effective quality" figure cited in §8.6 (sourced to [27]) describe *aggregate* quality over routine, broadly-saturated benchmarks. Both figures should be read as estimates of the gap on everyday tasks, **not** as representative of the gap on complex reasoning, architectural design, or long-horizon agentic work, where the open-weight deficit is materially larger.

The aggregate figure understates the gap on hard tasks. On standard, near-saturated benchmarks (MMLU, MMLU-Pro, GPQA Diamond, SWE-bench Verified, AIME) the open-weight frontier sits within roughly 3 points of the closed frontier [75, 77]. These benchmarks, however, are precisely the ones on which *both* classes of model perform well; a small spread on an easy

distribution does not imply a small spread on a hard one. As benchmarks become more discriminating, the gap widens:

Benchmark (task class)	Closed frontier (typical)	Open-weight frontier (typical)	Gap
GPQA Diamond — graduate-level science	~87% (high-80s)	~70% (high-60s–mid-70s)	~15–20 pp [75, 78]
SWE-bench Verified — real GitHub fixes	Higher; clearest on 15 min–1 h tasks	Lower (neutral bash-only harness)	Closed > open [79]
ARC-AGI-2 — abstract reasoning	~77–84%	No competitive public score	Large [76, 78]
Humanity's Last Exam (HLE)	~44–45%	~34–36%	~8–11 pp [77]
Terminal-Bench Hard — agentic coding	~61%	~43–46%	~15–18 pp [77]
SWE-bench Pro — held-out coding	~23% (frontier)	Not yet competitive	Large [78]

Note: Absolute SWE-bench and leaderboard figures are harness- and partly self-report-dependent and fluctuate as models are released; the table reports the robust direction and approximate magnitude, not a single point estimate.

The gap reopened in 2025 and is not monotonically closing. The Stanford HAI *AI Index 2026* finds that the open-weight performance gap *reopened* in 2025 after briefly closing in 2024: as of March 2026 the top closed-weight model leads the top open-weight model by 3.3% on the Chatbot Arena (up from 0.5% in August 2024), and six of the top ten Arena models are now closed-weight [75] *(post-cutoff source; published 13 April 2026, data as of March 2026)*. The gap therefore behaves as a fluctuating frontier race, not a one-directional convergence — and on contamination-resistant benchmarks the effective gap is roughly 1.7–1.9× the public-benchmark gap [80]. On ARC-AGI specifically, the arXiv:2603.13372 living survey documents a sharp "compositional cliff" from ARC-AGI-1 (frontier >96%) to ARC-AGI-2 (frontier 68–85%), a difficulty transition on which open-weight models have not yet posted competitive scores [76].

Provenance of the headline figures. The "85–95% of GPT-4" figure [9] originates from a peer-reviewed IEEE BigData 2025 cost-benefit analysis whose quality comparison is an aggregate over routine deployment tasks and is not stratified by task difficulty. The "98% of GPT-4 effective quality" figure [27] is sourced to a non-peer-reviewed commercial blog (localaimaster.com) that self-describes its content as "partially AI-assisted"; it should be treated as a vendor marketing claim rather than a measured benchmark result.

Recommendation. Treat the open-vs-cloud capability gap as *task-dependent*, not as a single percentage. For routine, non-architectural work the gap is small enough that local/open-weight deployment is a defensible default on cost and data-residency grounds. For architectural, long-horizon, or correctness-critical work, the closed frontier retains a load-bearing lead, and a single aggregate figure should not be used to justify routing such work to a local 70B-class model. The routing implications are reconciled in §8.6 (revised decision tree below).

8.6 The Emerging Best Practice: Hybrid Architecture

No single paradigm is universally optimal. The emerging best practice, reported in a commercial survey of 214 enterprise implementations *(localaimaster.com, 2025 — a commercial blog survey, not peer-reviewed research; results should be interpreted with appropriate caution regarding self-selection bias)*, is a **hybrid architecture**:

```
Query Classification Engine (unified: capability × data-residency)
 Step 1 — Classify along TWO axes:
  (a) Data sensitivity : Sensitive/Security-Critical  vs  Routine/Non-Sensitive
  (b) Capability demand: Architectural/Complex        vs  Routine
 Step 2 — Route by class:
  ├── Sensitive / Security-Critical  (any capability level)
  │    Default: Local / air-gapped LLM (Thick Client)
  │    Trade-off: data-residency MAXIMIZED ; capability SACRIFICED
  │    Exception: if ALSO Architectural/Complex AND a governed-cloud TEE path
  │     with enforceable ZDR is available, a documented risk-acceptance
  │     decision may route to governed cloud — never by default, only with
  │     explicit sign-off.  [§8.4 Secure Channel caveats apply]
  ├── Routine / Non-Sensitive
  │    Default: Cloud LLM (Secure Channel: mTLS + ZDR + DLP proxy)
  │    Trade-off: capability & cost OPTIMIZED ; data-residency LOW (acceptable)
  │    Local small model (8B–14B) only where latency/egress constraints dominate
  └── Architectural / Complex  (non-sensitive)
       Default: Governed Cloud frontier model
       Trade-off: capability MAXIMIZED ; data-residency LOW
       Rationale: per §8.5.8 the closed frontier retains a load-bearing lead on
        hard tasks; the local-70B option is a capability-for-residency trade,
        NOT an equal alternative.
       Fallback to Local Large Model (70B+) ONLY when data-residency is the
        binding constraint AND the task's capability bar is met by an
        open-weight 70B-class model (verify via task-specific eval, not the
        aggregate "85–95% of GPT-4" figure).
```

Figure 11: Hybrid architecture query classification and routing model (Section 8.6).

Capability vs. data-residency trade-off (see §8.5.8): For complex/architectural tasks, the local-70B option trades capability for data-residency; the governed-cloud branch should be the default when capability is the priority. The "Local Large Model (70B+) **or** Governed Cloud" branch above should therefore not be read as an equal choice — it is a conditional fallback. A local 70B-class model should be selected for Architectural/Complex work only when (a) data-residency is the binding constraint and (b) the task's capability bar has been verified against a task-specific evaluation, not against the aggregate "85–95% of GPT-4" figure.

Reported Benefits of Hybrid Architecture:

- **Significant cost reduction** vs. pure cloud deployment (at sufficient scale)
- **98% of GPT-4 effective quality** retention *(per [27], a non-peer-reviewed commercial blog; see §8.5.8 — this is an aggregate over routine tasks and overstates retention on complex/architectural work)*
- **Zero network latency** for local queries
- **~40% enterprise adoption rate** (localaimaster.com 2025 study)

Model Routing Strategy:

- Simple/routine queries → Local small models (8B–14B parameters)
- Complex/architectural queries → Cloud frontier models OR local large models (70B+)
- Sensitive queries → Always local/air-gapped, with Prevention for SCMs

---

9 Security Strategies and Implementation

9.1 The LLMSecOps Lifecycle

Security for AI-assisted development cannot be bolted on after deployment — it must be integrated throughout the entire lifecycle. The **LLMSecOps** framework, derived from the OWASP GenAI Security Solutions Landscape, maps security controls to each phase of the LLM development and operation lifecycle:

Lifecycle Phase	LLMOps Activities	LLMSecOps Controls
Scoping/Planning	Data suitability, model selection	Access control planning, compliance assessment, threat modeling
Data Augmentation	RAG, fine-tuning, RLHF	Data source validation, secure data handling, adversarial robustness testing

Lifecycle Phase	LLMOps Activities	LLMSecOps Controls
Development	Agent development, prompt engineering	SAST/DAST/IAST, LLM vulnerability scanning, secure coding practices
Test/Evaluation	Validation datasets, bias/fairness checks	Adversarial testing, penetration testing, agent scanning
Release	CI/CD pipelines, model packaging	AI/ML BOM, digital model signing, secure supply chain verification
Deploy	Infrastructure setup, API configuration	LLM WAF, MFA, network security validation, agent permission control
Operate	Feedback collection, model maintenance	LLM guardrails, incident detection/response, anomaly detection in agent chains
Monitor	Auto-retraining, model drift detection	Adversarial input detection, model behaviour analysis, immutable logs
Govern	Regular audits, data governance	Bias/fairness oversight, compliance management, agent action audit

9.2 API Interception and Data Loss Prevention (DLP) Proxies

Data Loss Prevention (DLP) proxies are the first line of defence for monitoring and controlling AI traffic leaving the corporate perimeter. They operate at Layer 7 (application layer) to inspect the actual content of prompts and responses, not just network metadata. The following implementation patterns describe how to deploy DLP controls for AI traffic.

Implementation: Deploy Layer 7 proxies (Envoy, Nginx, or dedicated AI security gateways) between developer endpoints and Cloud LLMs. Integrate regular expression and machine-learning-based DLP scanners to intercept, detect, and redact secrets, PII, and internal IP markers *before* the payload leaves the corporate boundary.

Capabilities required for code-aware DLP:

- **Entropy analysis** for secret detection (API keys, tokens, passwords)
- **Code structure understanding** — not just text pattern matching
- **AST-aware scanning** to detect proprietary algorithm patterns
- **Cross-file context awareness** to catch secrets referenced across files

Leading Solutions:

- Open Source: LLM Guard, LlamaFirewall, PromptGuard (Meta)

- Proprietary: Palo Alto Networks AI Runtime Security (AIRS), Cisco AI Runtime, F5 AI Gateway, Nightfall, Microsoft Purview

Enterprise AIRS Deployment Pattern (Real-World Implementation):

A production enterprise AI environment implements a two-firewall chain design for AI traffic inspection, providing both deep security and operational flexibility:

1. **First Inspection Layer:** Local firewalls (not connected to AIRS) — apply standard Next-Generation Firewall (NGFW) features including App-ID-based L7 inspection, antivirus, anti-spyware, DNS security, and vulnerability protection for known exploits
2. **Second Inspection Layer:** AIRS-integrated firewall — Prisma AIRS performs deep packet inspection of AI-specific traffic, analysing prompts and responses against:
   - Enterprise DLP signatures (personal data, IDs, proprietary code patterns)
   - AI security signatures (model poisoning, injection attacks, toxic content, database exploits)
   - Known attack techniques (prompt injection, model denial, poisoning)

Design Rationale: The two-firewall chain enables easy rollback to the original, non-AIRS topology if integration issues arise — a critical operational consideration for production environments where AI services must remain available.

Zone-Based Model Placement Strategy:

Enterprise deployments typically segment AI workloads into security zones based on data sensitivity:

Zone Type	Security Profile	Typical Workloads
AIRS Protected AI Workload Zone	Full prompt/response scanning, cloud telemetry enabled	Non-sensitive models, public-facing services
AI Workload Zone	Cloud analysis suppressed; L7 + on-prem security only; LiteLLM router present	Sensitive data models requiring local processing
Cloud-Based Models	S2S VPN tunnels; limited visibility due to SSL encryption requirements	Frontier models for complex/architectural queries

Operational Consideration: While micro-segmentation within zones provides maximum security, rapid environment changes in AI development can create significant administrative overhead. Some organisations opt for zone-level segmentation with robust firewall rules rather than fine-grained micro-segmentation, accepting a measured tradeoff between security granularity and operational agility.

Cloud Model Visibility Challenge: Traffic to cloud-based models presents inherent visibility limitations due to SSL/TLS encryption requirements. Organisations should actively explore enhancement options such as SSL decryption proxies, certificate pinning, or dedicated AI security gateways that can inspect encrypted AI traffic without breaking end-to-end security.

9.3 Zero Data Retention (ZDR) Enterprise Agreements

Zero Data Retention (ZDR) agreements are contractual clauses negotiated with cloud LLM providers that prohibit the provider from storing, logging, or using submitted prompts for model training. They address the data residualization risk identified in §2.2: even if transit is secure, submitted code may persist in provider infrastructure and be used for future fine-tuning. ZDR agreements are the primary contractual (as opposed to technical) control for cloud LLM deployments.

Implementation: Mandate strict contractual clauses with Cloud LLM providers (via Enterprise API tiers, not consumer chat interfaces) that guarantee payloads are ephemerally processed in memory and never logged, stored, or utilised for continuous pre-training or fine-tuning.

Key requirements for ZDR agreements:

- Explicit prohibition on using submitted code for model training
- Audit rights to verify compliance
- Data processing agreements compliant with GDPR Article 28
- Incident notification requirements for any data breach
- Geographic data processing restrictions (EU data stays in EU)

Verification challenge: ZDR agreements are contractual, not cryptographic. Organisations cannot technically verify that a provider has honored ZDR commitments. This is a fundamental limitation that only local deployment fully resolves. *(As established in §8.4.3, this limitation is inherent to the ZDR model; verifiable approaches such as zkUnlearner [44] (for verifiable machine unlearning, a related but distinct problem from ZDR verification) and TEE attestation remain research-stage only.)*

9.4 Air-Gapped / On-Premise Mandates for High-Sensitivity Workloads

For repositories classified as Confidential or Restricted, organisations should mandate the use of local, self-hosted LLMs that never transmit code outside the corporate perimeter. This section describes the policy framework and technical controls for enforcing on-premise-only AI development for high-sensitivity workloads.

Implementation: Establish organisational policies requiring the use of local, self-hosted LLMs for specific repositories containing core IP, cryptographic implementations, or compliance-heavy data (HIPAA/PCI/GDPR).

Policy framework:

- Classify repositories by sensitivity level (Public, Internal, Confidential, Restricted)
- Map sensitivity levels to permitted AI tools (Cloud LLM, Governed Cloud, Local Only, No AI)
- Enforce through IDE configuration management and network controls
- Audit compliance through centralised logging

9.5 Granular RAG Security Boundaries and Identity Propagation

RAG architectures introduce a critical security challenge: ensuring that the retrieved context respects the same access controls as the original data sources. Without proper identity propagation, a developer could use the LLM to bypass document-level permissions and retrieve sensitive information they are not authorised to access. The following implementation pattern describes how to enforce granular RAG security boundaries.

Implementation: Implement strict RBAC at the Vector Database level. The RAG retrieval pipeline must execute using the federated identity of the requesting developer (via OAuth2/OIDC token exchange), ensuring the LLM only retrieves and processes context the developer is explicitly authorised to view.

Technical controls:

- **Identity Propagation:** Developer's OIDC token passed through to Vector DB query
- **Row-Level Security:** Vector DB enforces document-level access control
- **Namespace Isolation:** Separate vector namespaces per project/classification level
- **Audit Logging:** All RAG retrievals logged with developer identity and retrieved document IDs
- **TTL on Embeddings:** Sensitive document embeddings expire and are re-indexed on access

9.6 Automated Code Scanning and Provenance Tracking

LLM-generated code introduces unique risks: hallucinated packages, license contamination from training data, and embedded vulnerabilities that traditional SAST tools may not detect.

This section describes how to extend existing code scanning pipelines to handle AI-generated code specifically, including provenance tracking and license compliance checks.

Implementation: Integrate SAST (Static Application Security Testing) and SCA (Software Composition Analysis) tools directly into the CI/CD pipeline and the IDE. Employ specialised provenance tools to scan LLM-generated code blocks against known open-source databases to detect unauthorised license contamination before the code is committed.

LLM-specific scanning requirements:

- **AI-generated code detection:** Flag code blocks likely generated by LLMs for enhanced review
- **License contamination scanning:** Check generated code against GPL/AGPL/LGPL databases
- **Secret scanning:** Detect hardcoded credentials in LLM-generated code
- **Vulnerability pattern detection:** SAST rules tuned for common LLM hallucination patterns (e.g., deprecated API usage, insecure defaults)

Solution landscape:

- Open Source: Garak.AI (LLM01), Modelscan (LLM01), CyberSecEval (Meta)
- Proprietary: Cisco AI Validation, Mend AI, Noma Security, Pillar Security

9.7 Zero-Trust Network Segmentation

Zero-Trust Network Segmentation is the infrastructure-layer control that enforces the security boundaries described in the preceding sections. By isolating AI workloads into dedicated network zones with strict egress controls, organisations can contain the blast radius of a compromised agent or misconfigured tool. The following implementation pattern describes how to architect zero-trust segmentation for AI environments.

Implementation: Architect the network such that on-premise LLM inference servers and Local GitLab instances reside in segmented VLANs/subnets with strict firewall rules, disconnected from environments with direct, unfiltered internet access.

Zero-Trust principles for AI environments:

- **Never trust, always verify:** Every AI interaction treated as an untrusted third-party event
- **Least privilege access:** Agents granted minimum permissions required for specific tasks
- **Micro-segmentation:** AI workload zones isolated from production systems
- **Continuous verification:** Agent identity and permissions re-verified on each action

- **Assume breach:** Design for containment, not just prevention

Practical Implementation Consideration:

While micro-segmentation provides maximum security granularity, real-world deployments must balance security with operational agility. In rapidly evolving AI development environments where infrastructure changes frequently, fine-grained micro-segmentation can create significant administrative overhead. Some organisations opt for zone-level segmentation (using 802.1Q VLANs) with robust L7 firewall rules based on application signatures (App-ID), accepting a measured tradeoff between security granularity and operational maintainability.

Recommended approach: Start with zone-based segmentation for core isolation, then apply micro-segmentation selectively to high-risk workloads where the security benefit justifies the operational cost.

9.8 Comprehensive Audit Logging and Telemetry

Audit logging is the foundation of incident response and compliance for AI-assisted development. Without complete, tamper-evident logs of all prompts, responses, tool calls, and agent actions, organisations cannot investigate security incidents or demonstrate regulatory compliance. This section describes the minimum telemetry requirements for AI security auditing.

Implementation: Implement centralised logging (Elasticsearch, Logstash, Kibana (ELK) stack, Splunk, or equivalent) of all prompts and responses. This enables forensic analysis in the event of a suspected breach, helps identify anomalous querying patterns, and satisfies regulatory compliance auditing.

Required telemetry for AI security:

- Developer identity (authenticated, not self-reported)
- Timestamp and session ID
- Model and version used
- Full prompt content (or hash for privacy-preserving logging)
- Response content (or hash)
- Files accessed by the agent
- Tool calls made by the agent
- External URLs accessed
- Data volume transmitted

Immutable log requirements: Logs must be tamper-evident (cryptographic chaining or Write Once Read Many (WORM) storage) to satisfy regulatory requirements and forensic integrity standards.

9.9 Agentic Security: Guardian Agent Pattern

The Guardian Agent Pattern introduces a secondary layer of AI-powered security that operates in real-time alongside the primary generative model. Rather than relying solely on static rules or human review, guardian agents use semantic analysis to detect and block sensitive data exfiltration, prompt injection attempts, and unauthorised tool usage. This section describes how to implement guardian agents as part of a defence-in-depth strategy.

Innovative Stream: Secondary AI agents act as **"Guardians"** for primary generative models. These agents perform real-time sanitization of prompts, removing PII and NDA-covered logic before the request reaches the external API.

Guardian Agent Functions:

- Real-time prompt sanitization (removing PII, NDA-covered logic, secrets)
- Capability boundary enforcement (blocking tool calls outside permitted scope)
- Anomaly detection in agent behaviour (flagging unusual file access patterns)
- Output validation (checking generated code for security vulnerabilities before delivery)

Implementation considerations:

- Guardian agents must themselves be secured against prompt injection
- Guardian agents should run in isolated execution environments
- Guardian agent decisions must be logged for audit purposes
- Human escalation path required for ambiguous cases

Failure-without-attack modes. A guardian agent can undermine the organisation without any adversary acting at all. *False positives*: a guardian calibrated too aggressively flags legitimate developer prompts as "sensitive," creating friction that pushes developers toward unsanctioned Shadow AI channels (§5.7.5) — the very uncontrolled data drain the guardian was deployed to close. *Availability impact*: because every prompt transits the guardian, guardian downtime, model unavailability, or miscalibration can block all AI-assisted development across the organisation, converting a security control into a single point of failure for developer productivity. *Recursive trust problem*: securing the guardian with another LLM-based guardian leads to an infinite regress, so the guardian must be protected by architectural means — sandboxed and local-only execution, deterministic rule pre-filtering, and least-privilege capability scoping — rather than by a further model that is itself susceptible to prompt

injection. *No empirical validation*: the Qwen3Guard deployment described in §16.7 (LL-9) reports no measured detection accuracy, false-positive rate, or latency overhead; the score=0.923 shown in the verification log is an illustrative example, not a measured result. Guardian agents should therefore be treated as a defence-in-depth supplement to — not a replacement for — ZDR agreements, prompt hygiene, and human review.

9.10 TRiSM Framework: Five-Pillar Governance

The **TRiSM (Trust, Risk, and Security Management) Framework for Agentic AI** [4] provides a comprehensive governance structure organised around five pillars:

Pillar 1: Explainability / Trust

- Chain-of-Thought (CoT) logging for all agent decisions
- Inter-agent traceability for multi-agent systems
- LIME/SHAP for agent decision explanation
- Decision-provenance graphs
- Standards: EU AI Act Arts. 13–14, NIST AI RMF, OECD AI Principles

Pillar 2: ModelOps (Lifecycle)

- Versioning and lineage for models and prompt configurations
- CI/CD safety gates (automated security testing before deployment)
- Rollout/rollback procedures for model updates
- Drift detection for model behaviour changes
- Standards: ISO/IEC 42001 (AIMS), ISO/IEC 42005, NIST AI RMF

Pillar 3: Application Security

- Prompt hygiene and secure prefixes
- Sandboxed tool execution environments
- Allowlisted actions (deny-by-default for agent capabilities)
- Prompt-injection pattern detection
- Standards: OWASP LLM Top-10, OECD Robustness

Pillar 4: Privacy and Data Protection

- Differential Privacy for training data
- Anonymization and pseudonymization of sensitive context

- Multi-Party Computation (MPC) for collaborative inference
- Homomorphic Encryption (HE/FHE) for encrypted inference (emerging)
- Trusted Execution Environments (TEEs) for hardware-isolated inference
- Standards: GDPR Art. 25/5/35, CCPA/CPRA, HIPAA §164

Pillar 5: Governance

- Human oversight mechanisms (human-in-the-loop for high-risk actions)
- Accountability structures (clear ownership of AI system decisions)
- Auditability (complete, tamper-evident logs)
- Data Protection Impact Assessments (DPIAs)
- Standards: EU AI Act Art. 14, NIST AI RMF, ISO/IEC 42001/42005

---

10 Regulatory and Compliance Framework

10.1 EU AI Act: The August 2026 Deadline

The EU AI Act represents the most significant regulatory development for AI-assisted software development. Its enforcement timeline creates urgent compliance requirements:

Date	Milestone
August 1, 2024	EU AI Act enters into force
August 2, 2025	Prohibited practices enforcement begins
August 2, 2026	**High-risk AI requirements effective**
February 2, 2027	Full enforcement across all provisions

Source: [29].

Finland became the first EU member state with fully operational AI Act enforcement powers on January 1, 2026. Other member states are implementing throughout Q1 2026.

Post-freeze regulatory development (26 March 2026). After this report's regulatory freeze date of 2026-03-21, the European Parliament adopted its negotiating position on the "digital omnibus" amendment to the AI Act on 26 March 2026, by 569 votes in favour, 45 against, and 23 abstentions [50]. The position would delay the application of high-risk AI obligations: to **2 December 2027** for high-risk systems specifically listed in the regulation (biometrics, critical infrastructure, education, employment, essential services, law enforcement, justice, and border

management); to **2 August 2028** for AI systems already covered by EU sectoral safety and market-surveillance legislation; and to **2 November 2026** for the Article 50 watermarking/labelling obligations on AI-generated content. This is a **Parliament negotiating position only — not final law**: interinstitutional (trilogue) negotiations with the Council must conclude before any change takes effect. Until an adopted amendment enters into force, the original timeline above (high-risk requirements effective **2 August 2026**) remains the operative legal default, and organisations should continue to plan against it. The episode confirms that the EU AI regulatory landscape remains in flux, and that deadlines cited in this section should be re-verified against official sources as the trilogue concludes [50].

Penalty Structure:

Violation Type	Maximum Penalty
Prohibited AI practices	**7% of global annual turnover** OR €35M
Supplying incorrect information	**3% of global annual turnover** OR €15M
Other violations	**1% of global annual turnover** OR €7.5M

Source: EU AI Act (Regulation (EU) 2024/1689).

High-Risk AI Requirements (effective August 2, 2026):

- Conformity assessments completed
- Technical documentation (Annex IV) finalized
- CE marking affixed
- Registration in EU database
- Human oversight mechanisms implemented
- Data governance and quality management systems

Relevance to Code Submission:

- AI systems used to generate code for "Critical Infrastructure" or "High-Risk Systems" require rigorous documentation of data used for inference
- The "Right to be Forgotten" (GDPR) is mathematically complex to enforce within neural network weights — encouraging Privacy-Preserving Inference techniques
- The EU's push for "Digital Sovereignty" aligns with the Thick Client strategy — keeping data within the EU avoids trans-Atlantic data transfer complexities (Schrems II implications)

Implication for organisations: Organisations that have not started conformity assessments by Q1 2026 are unlikely to meet the August 2, 2026 deadline. The 7% global turnover penalty is not a theoretical risk — Finland's enforcement authority is already operational. AI systems used to generate code for critical infrastructure require rigorous documentation and human oversight mechanisms.

10.1.1 EU General-Purpose AI (GPAI) Code of Practice

Alongside the binding AI Act timeline, the EU has established a **voluntary** compliance pathway for providers of general-purpose AI (GPAI) models: the **General-Purpose AI Code of Practice**, published on 10 July 2025 and confirmed by the European Commission and the AI Board as an adequate tool for providers to demonstrate compliance with their AI Act obligations (Articles 53 and 55) [51]. Although voluntary, adherence offers signatories reduced administrative burden and greater legal certainty than demonstrating compliance by other means.

The Code comprises three separately authored chapters:

- **Transparency** — applies to all GPAI model providers; operationalises the Article 53 documentation and training-data summary obligations, including a user-friendly Model Documentation Form and the AI Office's template for the "sufficiently detailed summary" of training data.
- **Copyright** — applies to all GPAI providers; requires a copyright-compliance policy under EU law, including recognition of text-and-data-mining (TDM) opt-outs (e.g., machine-readable opt-out declarations such as robots.txt / ai.txt).
- **Safety and Security** — applies only to providers of GPAI models with systemic risk (Article 55); covers systemic-risk assessment and mitigation, adversarial testing/red-teaming, serious-incident reporting, cybersecurity for model weights and infrastructure, and post-deployment monitoring.

As of the official EU signatory list, signatories include **Google, Microsoft, OpenAI, Anthropic, Amazon, Mistral AI, IBM, Cohere, Aleph Alpha, Black Forest Labs, and ServiceNow**, among others; **xAI** signed only the Safety and Security chapter and must demonstrate transparency and copyright compliance by alternative adequate means [51].

Implication for organisations: The Code is directly relevant to this report's scope because the models to which organisations submit sensitive source code and data are, in most cases, GPAI models whose providers are Code signatories. Provider-side transparency and copyright obligations govern how submitted material is documented and whether it may be used for training; the Safety and Security chapter addresses protection of the model weights and infrastructure that process submitted data. Organisations should (a) confirm whether a prospective provider is a Code signatory when evaluating legal certainty around data handling, (b) exercise copyright/TDM opt-outs where they do not want their code used for training, and

(c) recognise that the Code is **not** a "safe harbour" — it requires diligent implementation and does not by itself resolve the data-handling and zero-data-retention concerns central to this report [51].

10.2 NIST AI Risk Management Framework (AI RMF 1.0)

The NIST AI Risk Management Framework (AI RMF 1.0) provides a voluntary, consensus-driven framework for AI risk management organised around four core functions: GOVERN, MAP, MEASURE, and MANAGE. The following breakdown maps each function to specific requirements relevant to sensitive code submission and AI-assisted development.

GOVERN Function: Policies, processes, and accountability structures for AI risk management

- GOVERN 1: Policies and procedures for AI risk management
- GOVERN 2: Accountability structures and training
- GOVERN 6: Third-party software/data and supply chain policies

MAP Function: Context establishment and risk identification

- MAP 1: Context establishment and understanding
- MAP 4: Risk mapping for all components including third-party
- MAP 5: Impact characterisation for individuals, groups, communities

MEASURE Function: Risk quantification and tracking

- MEASURE 2: Trustworthy characteristic evaluation
- MEASURE 3: Risk tracking mechanisms

MANAGE Function: Risk treatment and monitoring

- MANAGE 1: Risk prioritisation based on MAP/MEASURE outputs
- MANAGE 3: Third-party risk management
- MANAGE 4: Risk treatment documentation and monitoring

Seven Trustworthiness Characteristics (NIST AI RMF):

1. Valid and Reliable
2. Safe
3. Secure and Resilient
4. Accountable and Transparent
5. Explainable and Interpretable

6. Privacy-Enhanced
7. Fair — with Harmful Bias Managed

Implication for organisations: The NIST AI RMF provides a voluntary framework, but its four functions (GOVERN, MAP, MEASURE, MANAGE) map directly to the controls needed for secure AI-assisted development. Organisations should adopt this framework as a baseline for their AI governance programs, particularly the GOVERN function which establishes accountability structures for AI risk management.

10.3 International Guidance Consensus

The NCSC/CISA *Guidelines for Secure AI System Development* represents an unprecedented international consensus, co-signed by cybersecurity agencies from 20+ countries including:

- UK NCSC, US CISA, NSA, FBI
- ACSC (Australia), CCCS (Canada), NCSC-NZ (New Zealand)
- ANSSI (France), BSI (Germany), NISC (Japan)

Four Key Lifecycle Areas:

1. **Secure Design:** Threat modeling, security + functionality balance
2. **Secure Development:** Supply chain security, asset protection, documentation
3. **Secure Deployment:** Infrastructure security, continuous model protection
4. **Secure Operation and Maintenance:** Behaviour monitoring, input monitoring, updates

Implication for organisations: The international consensus represented by the NCSC/CISA Guidelines means that secure AI development is no longer optional — it is becoming a regulatory and procurement requirement. Organisations should adopt the four lifecycle areas (Secure Design, Development, Deployment, Operation) as a baseline for their AI security programs.

10.4 ISO/IEC Standards

The following ISO/IEC standards provide international consensus on AI governance, risk management, and quality requirements. These standards are increasingly being referenced in regulatory frameworks (including the EU AI Act) and enterprise procurement requirements. The table below summarises the relevant standards and their current adoption status.

Standard	Scope	Adoption Status
ISO/IEC 42001:2023 (AIMS)	AI Management System certification	Early adoption by AWS, Anthropic, Google

Standard	Scope	Adoption Status
ISO/IEC 23894:2023	AI Risk Management guidance	Increased interest since Sep 2023; finance/healthcare prioritising
ISO/IEC 5259	Data quality and lineage standards	Emerging adoption

Source: ISO/IEC (42001:2023, 23894:2023, 5259).

Implication for organisations: ISO/IEC standards are increasingly referenced in regulatory frameworks and enterprise procurement. Organisations should plan for ISO/IEC 42001 (AIMS) certification as part of their AI governance strategy, particularly if they operate in regulated industries or supply to enterprises that require AI management system certification.

10.5 Regulatory Framework Mapping for Prompt Injection

Framework	Coverage	Requirements
OWASP LLM Top 10	LLM01	Input validation, sanitization
MITRE ATLAS	Multiple Technique IDs (TIDs)	Agent security controls
NIST AI RMF	MEASURE/MANAGE	Risk assessment requirements
EU AI Act	High-risk systems	Technical documentation, human oversight
ISO/IEC 42001	Clause 8	Operational planning
GDPR	Article 25	Data protection by design
NIS2	Article 14	Incident reporting

Source: Authors' analysis based on [11], [12], [16], [17].

10.6 OWASP COMPASS Methodology

The OWASP GenAI COMPASS RunBook provides a structured methodology for AI security assessment using the **OODA loop** (Observe, Orient, Decide, Act):

- **Observe:** Identify AI system components, data flows, and threat actors
- **Orient:** Map threats to OWASP LLM Top 10 and ASI Top 10 categories
- **Decide:** Prioritise risks using 5-point scoring system (Likelihood × Impact)
- **Act:** Implement controls, validate effectiveness, iterate

CWE/CVSS mapping provides standardised vulnerability assessment compatible with existing security tooling and regulatory reporting requirements.

---

11 Enterprise Deployment Patterns and Economics

11.1 The Coding-Agent Product Landscape (2026)

The 2025–2026 transition documented in §4.5 — from passive autocomplete to fully autonomous agentic assistants — has produced a fast-growing and bifurcated product market. Two classes now coexist, and the class distinction has direct security consequences for sensitive-code submission:

1. **Proprietary, cloud-mediated agents** (Cursor, GitHub Copilot, OpenAI Codex, Windsurf/Devin Desktop, Anthropic Claude Code) whose guardrails are internal and whose data paths are, by default, routed through vendor backends. Their controls are *declared* in documentation but their enforcement is *opaque* and unverifiable by the operator (see §7.10).
2. **Open-source, model-agnostic agents** (OpenCode, Cline, RooCode) whose control implementation is inspectable and patchable, and which can be operated with local models and no mandatory cloud round-trip — the condition that makes the mitigation paradigm of §8.5 (thick client / localised inference) and §9.4 (air-gapped mandates) operationally achievable.

Table 11.1.1 summarises the principal tools across the dimensions that matter for sensitive-code governance. "Auditable" denotes the operator's ability to inspect the *enforcement* of the tool's security controls (not merely read a vendor description of them).

Table 11.1.1 — Coding-agent tool comparison (2026)

Tool	Type	License	Cloud / Local	Auditable	Key Security Considerations
OpenCode	Open-source	MIT	Both — any provider incl. local; air-gapped capable	**Yes** (full source)	Stores no code/context data; model-agnostic (75+ providers); bring-your-own-key; local/air-gapped operation removes the cloud round-trip; MIT licence permits inspection,

Tool	Type	License	Cloud / Local	Auditable	Key Security Considerations
					modification, self-hosting. *Open-source case study — see §11.1.1.*
Cline	Open-source	Apache 2.0	Both — any provider incl. local (Ollama / LM Studio / OpenAI-compatible)	**Yes** (full source)	Plan-and-Act with per-step approval or auto-approve; BYO key/weights; hosted ToS applies only to the optional Cline API provider, not self-hosted use; MCP/SDK extensible. Actively maintained (not deprecated). *See §11.1.2.*
RooCode	Open-source	Open-source (permissive) [1]	Both — any provider incl. local	**Yes** (full source)	.rooignore file-system scoping (LL-4); client tool restriction is a *blacklist* (disabledTools) with no client-side allowlist yet (issue #11259) — enforce allowlists at the LiteLLM gateway; "Always Allow" trap (LL-1).
Claude Code	Proprietary	Proprietary (Anthropic)	Both — local CLI + cloud	**Limited**	Autonomous file read / command execution / code changes [70]; permission modes, sandbox, and hooks are configurable but agent-side; CVE-2025-55284 (DNS exfiltration), CVE-2025-59536 / CVE-2026-21852 (RCE / token

Tool	Type	License	Cloud / Local	Auditable	Key Security Considerations
					exfiltration via project files).
Cursor	Proprietary	Proprietary (Anysphere)	Cloud (backend-routed) + local	**Limited**	All requests route through Cursor backend; codebase indexing uploads code; Privacy Mode OFF → prompts to model providers + training [69]; .cursorignore does not bind terminal/MCP tools [68]; Grep-tool bypass reported.
GitHub Copilot	Proprietary	Proprietary (Microsoft / GitHub)	Cloud	**Limited**	Implicit multi-source context aggregation; 99.4% code-based jailbreak ASR (AAAI-25 [1]); RoguePilot passive prompt injection exfiltrates GITHUB_TOKEN [19]; training-data extraction demonstrated.
OpenAI Codex	Proprietary	Proprietary (OpenAI)	Both — local OS sandbox + cloud containers	**Limited** (enterprise Compliance API)	Cloud tasks in isolated OpenAI containers; network off by default; training off by default for Business/Enterprise/Edu; in-app browser / full-CDP access can expose sensitive data; audit via Compliance API only.

Tool	Type	License	Cloud / Local	Auditable	Key Security Considerations
Windsurf / Devin Desktop	Proprietary	Proprietary (Cognition)	Both — local + cloud agents	**Limited**	Lineage: Codeium → Windsurf → Cognition acquisition → renamed Devin Desktop; multi-agent "command center" over ACP; cloud handoff of tasks; limited transparency of agent runtime and data handling.

[1] RooCode is treated as open-source throughout this report (§16, LL-4); the exact SPDX licence should be confirmed against its repository at insertion time.

11.1.1 OpenCode — the open-source case study

OpenCode is the clearest contemporary exemplar of the open-source, model-agnostic paradigm this report advocates for high-sensitivity work. It is **MIT-licensed**, with approximately **176,000 GitHub stars** as of June 2026 (the project homepage reports 160,000+ stars, 900 contributors, 13,000+ commits, and 7.5M monthly developers), a predominantly TypeScript codebase, and an active release cadence (latest release **v1.17.8, 2026-06-17**). It is available as a terminal interface, desktop application, and IDE extension, and connects to **75+ LLM providers** via Models.dev, including local backends — enabling fully air-gapped operation.

Three properties make OpenCode the preferred open-source case study for sensitive-code submission:

- **No mandatory cloud round-trip.** The homepage states that "OpenCode does not store any of your code or context data, so that it can operate in privacy sensitive environments" [71]. Combined with local-model support, this realises the thick-client / air-gapped paradigms of §8.5 and §9.4 without the contractual dependence of a ZDR agreement (§9.3).
- **Full source auditability.** The MIT licence and public repository mean the *enforcement* of any declared control can be inspected and patched — precisely the property that proprietary agents cannot offer (§7.10). This is the operational meaning of the report's auditability stance: verifiability of controls, not merely their declaration.
- **Model agility without vendor lock-in.** Bring-your-own-key/model design means the tool layer stays constant while models are swapped underneath, and inference can be

redirected to a governed proxy or a local endpoint — the same LiteLLM routing pattern documented in §9.2 and §16.

OpenCode is therefore presented not as a universal recommendation but as the reference architecture for the "open-source + local-model + auditable" end of the spectrum: the configuration that minimises sensitive-code exposure while remaining inspectable. Its limitations — terminal-native UX, reliance on the operator to configure providers and proxy routing, and the same MCP supply-chain risks that affect all agents (§7.3, LL-6) — are the same limitations the rest of this report addresses through governance and proxy controls.

Security caveat: KiloCode, an MIT-licensed fork of OpenCode, demonstrates that "no mandatory cloud round-trip" is not guaranteed: an unconfigured hidden title agent can egress the user's first prompt — including attached file contents — to a cloud "small" model via a hardcoded fallback chain, even when the primary agent is configured for local inference or ZDR (see §4.6.8 for full analysis) [91].

11.1.2 Cline — active open-source (correcting the "morally outdated" characterisation)

Cline is **Apache 2.0**-licensed open-source with ~63.7k GitHub stars and 250+ contributors, available as an IDE extension, CLI, and SDK. Contrary to occasional characterisations of the project as "morally outdated," it is **actively maintained**: the repository ships a high release cadence with releases continuing through June 2026, and the project publishes current engineering content dated April 2026 [72]. Cline supports Plan-and-Act workflows (plan, then execute with per-step approval or auto-approve), every major model plus local backends (Ollama / LM Studio / any OpenAI-compatible endpoint) with bring-your-own-key or bring-your-own-weights, and MCP/SDK extensibility for custom tools and lifecycle hooks.

An important licensing nuance, clarified by the maintainers: the Cline Terms of Service apply **only** to users of the optional hosted "Cline" API provider. Users who supply their own API key or model endpoint are governed solely by the Apache 2.0 licence, and "Cline does not receive or store your input tokens, output tokens, underlying code, or other User Content" in that mode [72]. This makes self-hosted / BYO-key Cline directly comparable to OpenCode and RooCode on the open-source axis, and a valid component of the governed, auditable toolchain this report recommends.

11.1.3 Codex and Windsurf/Devin Desktop — proprietary, limited auditability

OpenAI Codex [73] is a proprietary coding agent available across Free, Plus, Pro, Business, Edu, and Enterprise ChatGPT plans, delivered as an app, CLI, IDE extension, and web client. Codex Cloud tasks execute in isolated OpenAI-managed containers (with a two-phase setup/agent

runtime in which the agent phase is offline by default), while Codex Local (app/CLI/IDE) runs in an OS-enforced sandbox with network access off by default. Training on inputs/outputs is off by default for Business/Enterprise/Edu; consumer use follows ChatGPT training controls. Auditability is limited to the proprietary runtime, with enterprise visibility provided through the Compliance API (prompt text, responses, metadata, retained up to 30 days for ChatGPT-authenticated usage). Notable risk surfaces include the in-app browser with full Chrome DevTools Protocol access, which the documentation itself warns "may put your data at risk."

Windsurf / Devin Desktop [74] is proprietary. The product lineage is Codeium → Windsurf → acquired by Cognition → renamed **Devin Desktop**; the current page is titled "Devin Desktop" and states "Windsurf is now Devin Desktop." It is positioned as a multi-agent "command center" (Spaces, Kanban, fleet management of local and cloud agents) over the Agent Client Protocol (ACP), with a free world-class model (SWE-1.6) and cloud handoff of long-running tasks. As a proprietary, cloud-capable agent whose runtime and data-handling internals are not published, it offers limited auditability and falls on the same side of the §7.10 opacity argument as Cursor and Copilot: controls are declared, not inspectable.

For both tools, the report's general guidance applies: route traffic through a DLP proxy (§9.2), enforce tool allowlists at the gateway (§13.6), restrict to local/sandboxed execution for sensitive work (§9.4), and treat any cloud delegation of a Confidential/Restricted repository as requiring a ZDR agreement (§9.3) plus network egress controls.

11.2 Current Enterprise Adoption Landscape (2025)

Deployment Pattern	Adoption Rate	Description
Hybrid Architecture	~40%	Local models (85% queries) + Cloud APIs (15% edge cases)
Cloud-Only Enterprise	35%	Azure OpenAI, AWS Bedrock, Google Vertex AI
Local-Only	15%	On-premise H100/H200 clusters, air-gapped environments
Shadow AI	~27% (untracked)	Unsanctioned personal model usage by developers

Source: Menlo Ventures 2025 State of Generative AI in the Enterprise; localaimaster.com (214 enterprise study; per [27], a non-peer-reviewed commercial blog; see §8.5.8 for caveats)

Critical finding: The approximately 27% Shadow AI figure means that for every 3–4 sanctioned AI interactions, there is approximately 1 unsanctioned interaction occurring outside all security controls. This represents the largest uncontrolled data leakage vector in most organisations.

11.3 Enterprise Security Concern Rankings (2025)

For the first time in recorded history, **AI/LLMs ranked as the #1 cybersecurity concern**, surpassing ransomware:

Concern	Percentage	Year Over Year
AI, LLMs, and privacy risks	**29%**	New #1
Ransomware, malware, data extortion	21%	Was #1
Cloud security misconfigurations	15%	—
Supply chain attacks	12%	—

Source: Arctic Wolf 2025 Trends Report, survey of 1,200+ senior IT and cybersecurity decision-makers across 15 countries

11.4 World Economic Forum Cybersecurity Outlook 2026

The WEF Global Cybersecurity Outlook 2026 [25] (survey of 804 participants from 92 countries, August–October 2025) found:

Finding	Percentage
AI-related vulnerabilities as fastest-growing cyber risk	**87%**
Concern about data exposure to AI	**34%** (up from 22% in 2025)
Third-party/supply chain as greatest resilience challenge	**78%** (highly resilient organisations)

Source: World Economic Forum Global Cybersecurity Outlook 2026; verified by Computer Weekly, January 2026

11.5 Enterprise AI Platform Security Features

The following table compares the security features offered by major enterprise AI platforms. This comparison is intended to help organisations evaluate platform options based on their specific security and compliance requirements. Note that feature availability may vary by subscription tier and region.

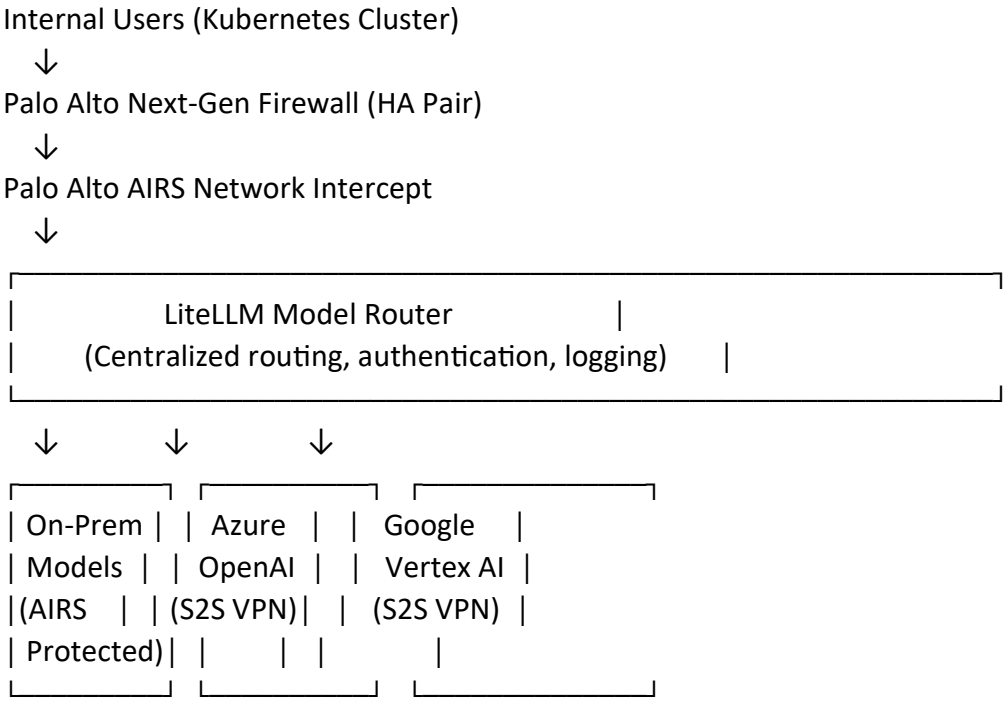
Platform	Key Strengths	Security Features
Microsoft Azure AI	Integration, enterprise scale	Private networking, Azure AD, content filtering
AWS Bedrock	Model diversity, scalability	VPC isolation, IAM integration
Google Vertex AI	Long-context, Workspace integration	VPC Service Controls, DLP APIs

Platform	Key Strengths	Security Features
OpenAI Enterprise	Frontier capability	Zero data retention option, team controls

Source: Authors' analysis of vendor documentation.

Real-World Deployment Topology Example:

A production enterprise AI environment demonstrates how multiple platforms can be integrated under a unified security architecture:



Key architectural patterns observed in production:

1. **Centralised Model Router (LiteLLM):** Provides unified authentication, rate limiting, and logging across all model providers — single control point for policy enforcement
2. **Dual-Path Architecture:** On-prem models for sensitive workloads (full AIRS inspection) + cloud models for frontier capabilities (S2S VPN tunnels with limited visibility due to SSL encryption)
3. **Cloud-Delivered Telemetry:** AIRS provides cloud-based traffic analysis and policy updates while respecting data residency requirements through region selection options
4. **Frontend Separation:** Distinct zones for public-facing AI services (Nginx + AnythingLLM) vs. internal secure access, preventing cross-contamination of contexts

11.6 The Agentic AI Security Race (2026)

Enterprise AI assistants are now deeply integrated into core business operations:

- Connected to ticketing systems and source code repositories
- Integrated with chat platforms and cloud dashboards
- Capable of opening pull requests autonomously
- Querying internal databases and booking services
- Triggering workflows and CI/CD pipelines

The Cisco AI Readiness Index 2025 [46] describes this level of access as agentic AI systems that "take action by navigating tasks, workflows, and even business decisions autonomously" — with 83% of organisations planning to deploy AI agents. Dark Reading's 2026 reader survey identified **agentic AI attacks** as the most likely trending security reality for 2026.

Key statistics:

- **~97 million** (reported) monthly MCP SDK downloads (February 2026)
- **10,000+** (reported) active tool servers connected to agent frameworks
- Security research documenting "accelerating catalogue of exploits that remain fundamentally unsolved"

11.7 Implementation Priority Matrix

Priority	Action	Rationale	Timeline
P0 (Immediate)	Audit current AI usage (Shadow AI detection)	Cannot protect what you cannot see	Week 1
P0 (Immediate)	Audit coding assistant configurations for RoguePilot-style vulnerabilities	Critical CVSS 9.6 vulnerability	Week 1
P1 (Short-term)	Implement Prevention for SCMs	Low cost, high impact	Month 1
P1 (Short-term)	Deploy DLP proxy for cloud LLM traffic	Immediate data leakage reduction	Month 1
P2 (Medium-term)	Evaluate Thick Client TCO break-even	18-month horizon now viable	Month 3
P2 (Medium-term)	Implement ZDR enterprise agreements	Contractual protection for cloud usage	Month 3

Priority	Action	Rationale	Timeline
P3 (Long-term)	Begin EU AI Act conformity assessment	August 2026 deadline	Month 6
P4 (Strategic)	Plan FHE/TEE integration	Emerging technology watch	12–24 months

Source: Authors' synthesis based on §8–§13.

---

12 Emerging Technologies and Future Directions

12.1 Fully Homomorphic Encryption: The Path to Production

FHE represents the long-term solution to the privacy-utility paradox: inference directly over ciphertexts, with confidentiality guaranteed purely by cryptography, without requiring trust in any hardware or provider. The 2025–2026 research breakthroughs have significantly accelerated the timeline to production viability.

Cross-reference: The full FHE research trajectory — including the four breakthrough papers (EncryptedLLM, Hyperion, ELLMo, Orion), the practicality assessment table, and the provenance note — is documented in §3.3. The summary below highlights only the strategic implications for future planning.

Practical Envelope (March 2026) — see §3.3 for full detail:

- **Now viable:** Token sampling (Hyperion), small models (<1B params)
- **6–18 months:** Medium models (1–10B params) in research/prototype
- **18–36 months:** Large models (>10B params) approaching viability
- **Not yet viable:** Real-time inference for large autoregressive models

Companies advancing FHE:

- **CipherSonic Labs:** GPU/FPGA acceleration, optimised packing schemes
- **Duality:** Hardware abstraction layer for OpenFHE, ASIC optimisation

Long-term significance: FHE would allow organisations to use cloud LLMs for sensitive code without any trust in the provider — the provider processes only ciphertext and never sees the plaintext code. This would fundamentally resolve the Trust Gap described in §2.2.

12.2 Trusted Execution Environments: Mature but Imperfect

TEEs (Intel SGX, AMD SEV, ARM TrustZone) offer a more immediately practical alternative to FHE. They create hardware-isolated enclaves where LLM inference occurs without exposing data to the host OS or cloud provider.

Cross-reference: For the full technical analysis of TEE limitations — including the TEE.Fail physical attack (DDR5 memory bus interposition, <$1,000 hardware), cache side-channel attacks (Flush+Reload: 99.2%, Prime+Probe: 53.4% AES key recovery), power analysis attacks, +4–10% throughput overhead, and the 30B parameter memory constraint — see §3.4.

Current limitations summary (2026) — see §3.4 for full detail:

- Side-channel attacks against SGX have been documented (TEE.Fail, Spectre, Meltdown variants) [36]; cache attacks show technique-dependent success (Flush+Reload: 99.2%, Prime+Probe: 53.4%) [37a]
- Hardware trust assumption introduces supply chain risks
- Memory limitations constrain model size to ~30B parameters in enclaves [38]
- Attestation protocols add latency; **+4–10% throughput overhead** [38]

"Trusted Execution Environments (TEEs) were once the crown jewel of confidential computing... But recent attacks have exposed fundamental weaknesses." — Wodan AI, October 2025

Recommendation: TEEs are currently the most practical option for privacy-preserving cloud inference but should be combined with other controls (ZDR agreements, DLP proxies) rather than relied upon as a sole defence.

12.3 Agentic Security: The Guardian Pattern at Scale

The convergence of Zero-Trust Architecture (ZTA) and AI represents the next frontier. By treating every AI interaction as an untrusted third-party event, organisations can implement **Inline AI Runtime Security** — sub-millisecond inspection of every token exchanged.

Next-Generation AI Runtime Security Capabilities (2026):

Capability	Description	Maturity
Sub-millisecond token inspection	Real-time prompt/response analysis	Production
Agent behaviour baselining	Learning normal agent patterns	Emerging
Cross-agent correlation	Detecting coordinated attacks across agent populations	Research

Capability	Description	Maturity
Autonomous threat response	AI defending against AI	Emerging

Market Leaders:

- Palo Alto Networks Prisma AIRS
- Cisco AI Runtime Security
- F5 AI Gateway
- Microsoft Defender for AI

12.4 Synthetic Twin Generation

Concept: Proprietary codebases are transformed into statistically identical but functionally distinct "Twins" for model fine-tuning. The twin preserves the structural and statistical properties of the original codebase (enabling effective fine-tuning) while replacing the actual proprietary logic with functionally equivalent but non-sensitive alternatives.

Benefit: Enables model customization without exposing actual proprietary logic to the fine-tuning process or the resulting model's weights.

Current status: Active research area; no production-ready tooling as of March 2026.

12.5 Emerging Adversarial Threats

Adversarial Coordination: Multiple agents circumventing safeguards to optimise shared objectives. Requires detection of coordinated attack patterns across agent populations — a capability that does not yet exist in production security tools.

Self-Amplification and Self-Modifying AI: Cascade failures with limited human intervention windows. Requires kill-switches and circuit breakers at the infrastructure level.

Self-Replication via Provisioning APIs: Autonomous agent proliferation through cloud infrastructure APIs. Requires registration controls and versioning policies for agent deployments.

Manipulative Social Engineering by AI: Exploiting human biases through automated psychological attacks at scale. Requires human-factors UI safeguards and trust calibration mechanisms.

Implication for organisations: These threats have no current defences. Organisations should treat them as a watch list and ensure their incident response plans include scenarios for each. The lack of production-ready detection tools means human oversight and infrastructure-level controls (kill-switches, circuit breakers) are the only available mitigations.

12.6 Research Gaps and Open Problems

Fundamental unsolved problems (March 2026):

1. **No Defence Against Semantic Attacks:** Current defences cannot distinguish between legitimate and malicious semantic content at the L3 layer
2. **Autonomous Agent Accountability:** No framework for attributing agent actions to human operators in legally defensible ways
3. **Cross-Platform Threat Intelligence:** No shared database of AI-specific threats and exploits across vendors
4. **Real-Time Encrypted Inference:** FHE still too slow for production large-model inference
5. **Agent-to-Agent Security:** No standardised security protocols for inter-agent communication
6. **Code-Specific Safety Alignment:** RLHF training on natural language does not transfer to code-based attack prevention

---

13 Recommendations and Decision Framework

13.1 Approach Selection Framework

Organisation Profile	Recommended Primary Approach	Secondary Approach	Rationale
High Security, Budget-Constrained	Prevention + Secure Channel	Selective Obfuscation	Maximum security within budget
High Security, Adequate Budget	Thick Client (Local)	Hybrid with Cloud for non-sensitive	Gold standard; AI-mediated attacks prevented
Medium Security, Rapid Deployment	Secure Channel + Prevention	Obfuscation for critical modules	Quick wins with existing infrastructure
Distributed/Remote Workforce	Hybrid Architecture	Local for sensitive, cloud for collaboration	Balances security and usability

Organisation Profile	Recommended Primary Approach	Secondary Approach	Rationale
Regulated Industry (Finance/Health)	Thick Client (Air-Gapped)	Prevention for any cloud usage	Compliance requirements mandate local
Startup/SME	Secure Channel (ZDR) + Prevention	Shadow AI detection first	Cost-effective starting point

Source: Authors' synthesis based on §8 paradigms and §9 strategies.

13.2 Decision Tree for Sensitive Code Submission

Is the code classified as Security-Critical or NDA-covered?
├── YES → Is local deployment feasible (budget, technical capacity)?
│ ├── YES → Deploy Thick Client (Local LLM)
│ │ Apply Prevention for any residual cloud usage
│ └── NO → Apply Prevention (hide SCMs from context)
│ Use Secure Channel (ZDR agreement) for cloud
│ Deploy DLP proxy for all cloud traffic
└── NO → Is the code Internal (not public)?
├── YES → Use Secure Channel (ZDR agreement)
│ Apply Prevention as additional layer
│ Monitor with DLP proxy
└── NO → Standard cloud LLM usage acceptable
Maintain audit logging

Figure 12: Decision tree for sensitive code submission approach selection (Section 13.2).

13.3 Key Metrics for Decision Making

For Thick Client Viability:

- High monthly token volume → Local deployment economically favorable
- Adequate budget for upfront hardware investment → Hybrid or Local recommended
- Security classification: High/Critical → Thick Client mandatory
- Compliance requirement (GDPR/HIPAA/PCI) → Local strongly preferred

For Approach Effectiveness:

- Utility Loss < 15% → Obfuscation viable as supplementary layer
- Module coupling < Medium → Prevention viable as primary approach

- Network latency tolerance > 200ms → Cloud acceptable for non-sensitive queries

13.4 Immediate Action Items (P0 — Week 1)

1. **Shadow AI Audit:** Deploy network monitoring to identify all AI tool usage, sanctioned and unsanctioned. Cannot protect what you cannot see.
2. **RoguePilot Vulnerability Assessment:** Audit all GitHub Copilot and similar coding assistant configurations for passive prompt injection vulnerabilities. Check all repositories opened in Codespaces environments.
3. **Credential Exposure Scan:** Scan all repositories and IDE configurations for hardcoded secrets that may have been exposed to cloud LLMs through context window aggregation.
4. **AI Tool Inventory:** Catalogue all AI tools in use, their data retention policies, and whether enterprise ZDR agreements are in place.

13.5 Security Solutions Reference

The OWASP GenAI Security Solutions Landscape maps 80+ vendors and open-source solutions across the LLMSecOps lifecycle. The following table provides a curated selection of solutions for each stage, organised by open-source and proprietary categories. This reference is intended to help organisations build a comprehensive security toolchain for AI-assisted development. Use this as a starting point for evaluating security solutions that match your organisation's specific requirements and budget constraints.

80+ vendors mapped across the LLMSecOps lifecycle (OWASP GenAI Security Solutions Landscape):

Lifecycle Stage	Open Source Solutions	Proprietary Solutions
Threat Modeling	StrideGPT	MitreAtlas, Seezo, PILLAR
Vulnerability Scanning	Garak.AI, Modelscan, CyberSecEval	Cisco AI Validation, Mend AI, Noma Security
Guardrails/Firewalls	LLM Guard, LlamaFirewall, PromptGuard	Palo Alto AIRS, Cisco AI Runtime, F5 AI Gateway
Red Teaming	Prompt Foo, DeepTeam	Adversa AI, Enkrypt AI, Mindgard, SplxAI
Posture Management	—	Prisma Cloud AI-SPM, Microsoft Defender AI-SPM

Lifecycle Stage	Open Source Solutions	Proprietary Solutions
Agentic-Specific	Agentic Radar, MCP Secure Gateway	Cortex Cloud AI-SPM, Noma, Zenity, Enkrypt AI

Source: [13].

13.6 MCP Governance Workflow

For organisations using MCP-based agentic coding assistants, the OWASP Practical Guide for Securely Using Third-Party MCP Servers (October 2025) recommends the following governance workflow:

MCP Server Submission
→ Automated Scanning (tool description analysis, hash verification)
→ Security Review (manual review for high-risk servers)
→ Approved Registry (internal catalog of vetted MCP servers)
→ Deployment (with pinned versions and integrity monitoring)
→ Re-validation (triggered by version updates or security advisories)

Authentication requirements: OAuth 2.1, OIDC/PKCE for all MCP server connections. No anonymous or API-key-only authentication for production deployments.

---

14 Conclusion

14.1 The Fundamental Tension

The integration of Large Language Models into enterprise software development has created a fundamental tension that cannot be resolved by any single technical control: **the more context an LLM has, the more useful it is — and the more sensitive data it is exposed to**. This report has documented the empirical evidence that this tension is being actively exploited, with attack success rates of 99.4% against the world's most widely deployed coding assistant (GitHub Copilot) and zero-click data exfiltration demonstrated in production systems.

14.2 The Evolving Threat Landscape

The threat landscape has evolved from theoretical to actively exploited in the span of 12 months. The five incidents documented in §2.4 — EchoLeak (CVE-2025-32711, May 2025), CVE-2025-55284 (June 2025), RoguePilot (February 2026, CVSS 9.6), CVE-2026-2256 (March 2026, CVSS 6.5), and the March 2026 web-based indirect prompt injection — represent the transition from theoretical risk to actively exploited vulnerability. See §2.4 for the full incident timeline.

The transition from theoretical to actively exploited is complete. Empirical evidence from the incidents documented in §2.4 indicates that similar systems are currently being exploited; organisations lacking these controls have a high likelihood of exposure.

14.3 The Viable Path Forward

This report's analysis leads to the following conclusions:

No single approach is sufficient. The four paradigms (Prevention, Obfuscation, Secure Channel, Thick Client) each address different aspects of the threat landscape. Obfuscation has been significantly downgraded in viability due to LLM de-obfuscation capabilities (§8.3). The other three approaches remain viable and should be combined.

The Thick Client approach has become economically viable. The break-even period has shortened significantly (now under 2 years for many organisations per IEEE BigData 2025). Open-source models now achieve 85–95% of GPT-4 quality *(an aggregate estimate derived from benchmark comparisons against current frontier models; [9] compares open-weight models to GPT-5 and Claude-4, not directly to GPT-4)*. For organisations with high-security requirements, local deployment is no longer a luxury — it is the only approach that prevents AI-mediated supply chain attacks.

The hybrid architecture is the emerging best practice. Local deployment for sensitive workloads (85% of queries), governed cloud access for non-sensitive tasks (15%), with Prevention/Isolation for security-critical modules in all contexts.

Regulatory compliance is now a forcing function. The EU AI Act's August 2, 2026 deadline for high-risk AI system compliance, with penalties up to 7% of global annual turnover, creates urgent pressure to implement governance structures that align with the Thick Client and Prevention approaches.

FHE represents the long-term solution. While not yet production-ready for large LLMs, the 2025–2026 research breakthroughs (EncryptedLLM, Hyperion, ELLMo) have significantly accelerated the timeline. Organisations should monitor FHE progress and plan for integration within 18–36 months.

14.4 Final Recommendations

1. **Treat AI security as a first-class security domain.** AI/LLMs are now the #1 cybersecurity concern (29%), surpassing ransomware (21%). Security budgets and organisational attention must reflect this reality.

2. **Implement the hybrid architecture.** Local deployment for sensitive workloads, governed cloud for non-sensitive, Prevention for security-critical modules in all contexts.

3. **Address Shadow AI immediately.** Approximately 27% of unsanctioned AI usage represents the largest uncontrolled data leakage vector. Detection and governance must precede all other controls.
4. **Prepare for the EU AI Act deadline.** August 2, 2026 is not a distant deadline — conformity assessments, technical documentation, and human oversight mechanisms take months to implement.
5. **Monitor the FHE research trajectory.** The technology is advancing rapidly. Organisations should establish a watch program and plan for integration as it becomes production-ready.
6. **Treat agentic AI as a new threat category.** The autonomous insider threat model, AI-mediated supply chain attacks, and cascading multi-agent failures require new detection and response capabilities that traditional security tools do not provide.

---

15 References

15.1 A.1 Peer-Reviewed Research

The following sources have undergone formal peer review at conferences with published proceedings or in peer-reviewed journals.

[1] W. Cheng, K. Sun, X. Zhang, and W. Wang, "Security Attacks on LLM-based Code Completion Tools," in *Proc. 39th AAAI Conf. Artificial Intelligence (AAAI-25)*, Philadelphia, PA, USA, Feb. 2025. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/34537 *(Key findings: 99.4% attack success rate on GitHub Copilot; 2,173 usernames and 54 email addresses extracted via prompt injection. Cited in: §6.1, §6.3.2, §11.1.)*

[2] P. Reddy and A. S. Gujral, "EchoLeak: The First Real-World Zero-Click Prompt Injection Exploit in a Production LLM System," in *Proc. AAAI Symposium Series*, vol. 7, no. 1, pp. 303–311, 2025, doi: 10.1609/aaaiss.v7i1.36899. [Online]. Available: https://arxiv.org/abs/2509.10540 *(CVE-2025-32711; CVSS 9.8 per NVD. Published at AAAI Fall Symposium Series 2025.) (Note: AAAI Fall Symposium Series is a non-archival workshop series with lighter peer review than main-track AAAI conferences. Cited in: §2.4, §5.3.)*

[4] S. Raza, R. Sapkota, M. Karkee, and C. Emmanouilidis, "TRiSM for Agentic AI: A Review of Trust, Risk, and Security Management in LLM-based Agentic Multi-Agent Systems," *AI Open*, vol. 7, pp. 71–95, 2026, doi: 10.1016/j.aiopen.2026.02.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651026000069 *(Five-pillar governance framework; CSS/TUE metrics. Cited in: §7.7, §9.10.)*

[5] R. Sapkota, K. I. Roumeliotis, and M. Karkee, "AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges," *Information Fusion*, 2025, doi: 10.1016/j.inffus.2025.103599. [Online]. Available: https://arxiv.org/abs/2505.10468 *(Architectural distinctions; multi-agent risk amplification. Cited in: §4.5, §7.1, §7.7.)*

[6] L. de Castro, D. Escudero, A. Agrawal, A. Polychroniadou, and M. Veloso, "EncryptedLLM: Privacy-Preserving Large Language Model Inference via GPU-Accelerated Fully Homomorphic Encryption," in *Proc. 42nd Int. Conf. Machine Learning (ICML 2025)*, vol. 267, 2025. [Online]. Available: https://proceedings.mlr.press/v267/de-castro25a.html *(200× GPU speedup over CPU FHE. Cited in: §3.3, §3.4.)*

[9] G. Pan, V. Chodnekar, A. Roy, and H. Wang, "A Cost-Benefit Analysis of On-Premise Large Language Model Deployment: Breaking Even with Commercial LLM Services," in *Proc. IEEE Int. Conf. Big Data (BigData 2025)*, 2025. [Online]. Available: https://arxiv.org/abs/2509.18101 *(Break-even: 6–24 months depending on model scale. Cited in: §8.5.4, §8.5.8, §14.3, §A.3.)*

[37a] F. Rauscher, C. Fiedler, A. Kogler, and D. Gruss, "A Systematic Evaluation of Novel and Existing Cache Side Channels," in *Proc. Network and Distributed System Security Symp. (NDSS 2025)*, Feb. 2025. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2025-253-paper.pdf *(Flush+Reload achieves 99.2% data recovery (13,000 encryptions); Prime+Probe achieves only 53.4% even with 1 M encryptions. Cited in: §3.4, §8.5.6, §12.2.)*

[37b] G. Le Gonidec, G. Bouffard, J.-C. Prevotet, and M. Mendez Real, "Do Not Trust Power Management: A Survey on Internal Energy-based Attacks Circumventing Trusted Execution Environments Security Properties," *ACM Trans. Embedd. Comput. Syst.*, vol. 24, no. 4, Art. 63, Jul. 2025, doi: 10.1145/3735556. [Online]. Available: https://hal.science/hal-05067675v1/document *(Survey of internal energy-based attacks on TEEs including CLKSCREW, VoltJockey (2019), and Plundervolt. Cited in: §3.4, §8.5.6, §12.2.)*

[38] M. Chrapek, M. Copik, E. Mettaz, and T. Hoefler, "Confidential LLM Inference: Performance and Cost Across CPU and GPU TEEs," in *Proc. 2025 IEEE Int. Symp. Workload Characterization (IISWC)*, Irvine, CA, USA, Oct. 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11241987 *(TDX/SGX overheads 4–10% (single socket), up to 23% (multi-socket); GPU TEEs (H100 cGPU) 4–8% overhead. Cited in: §3.4, §8.5.6.)*

[39] Z. Tian, B. Tian, J. Lv, Y. Chen, and L. Chen, "Enhancing Vulnerability Detection via AST Decomposition and Neural Sub-Tree Encoding," *Expert Systems with Applications*, vol. 238, Part B, Mar. 2024, doi: 10.1016/j.eswa.2023.121865. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423027173 *(AST + neural sub-tree encoding (TrVD): 92.85% precision, 83.97% recall, 88.19% F1-score on synthetic benchmarks; real-world (D2A) accuracy ≈64%. Cited in: §8.2.1.)*

[41] L. De Tomasi, C. Di Sipio, A. Di Marco, and P. T. Nguyen, "Simplicity by Obfuscation: Evaluating LLM-Driven Code Transformation with Semantic Elasticity," in *Proc. 29th Int. Conf. Evaluation and Assessment in Software Engineering (EASE 2025)*, Istanbul, Turkey, Jun. 2025, pp. 732–738. [Online]. Available: https://arxiv.org/abs/2504.14024 *(GPT-4-Turbo achieves 81% obfuscation pass rate with few-shot prompting but reduces cyclomatic complexity (average ΔCC ≈ −1.0 McCabe points with few-shot prompting). Cited in: §3.6, §8.3.2.)*

[42] G. Chen, X. Jin, and Z. Lin, "JsDeObsBench: Measuring and Benchmarking LLMs for JavaScript Deobfuscation," in *Proc. 2025 ACM SIGSAC Conf. Computer and Communications Security (CCS)*, Taipei, Taiwan, Oct. 2025, pp. 36–50, doi: 10.1145/3719027.3744871. [Online]. Available: https://arxiv.org/abs/2506.20170 *(GPT-4o achieves 95.99% syntax accuracy and 93.42% execution accuracy on JavaScript deobfuscation; 97.24% / 62.60% average across all models. Claude 3 was not evaluated. Cited in: §8.3.1.)*

[53] A. Shapira, P. A. Gandhi, E. Habler, and A. Shabtai, "Mind the Web: The Security of Web Use Agents," in *Proc. ACM Asia Conf. Computer and Communications Security (AsiaCCS)*, 2026, pp. 835–851, doi: 10.1145/3779208.3805968. [Online]. Available: https://arxiv.org/abs/2506.07153 *(Task-aligned injection achieves >80% ASR on OpenAI Operator and other web-use agents. Cited in: §7.8.)*

[54] Y. Gong, D. Ran, J. Liu, C. Wang, T. Cong, A. Wang, et al., "FigStep: Jailbreaking Large Vision-Language Models via Typographic Visual Prompts," in *Proc. AAAI Conf. Artificial Intelligence (AAAI-25)*, 2025 (Oral). [Online]. Available: https://arxiv.org/abs/2311.05608 *(82.5% mean ASR on 6 LVLMs via typographic image prompts; root cause is deficiency of safety alignment for visual embeddings. Cited in: §7.9.)*

[56] M. Chen, K. Wang, L. Lu, J. Zhang, and T. Zhang, "Hijacking Large Audio-Language Models via Context-Agnostic and Imperceptible Auditory Prompt Injection (AudioHijack)," in *Proc. IEEE Symp. Security and Privacy (S&P 2026)*, 2026. [Online]. Available: https://arxiv.org/abs/2604.14604 *(79%–96% success on 13 LALMs; commercial voice agents induced to perform unauthorised tool calls. Cited in: §7.9.)*

[66] V. S. Sadasivan, S. Feizi, R. Mathews, and L. Wang, "Attacker's Noise Can Manipulate Your Audio-based LLM in the Real World," in *Proc. 19th Conf. European Chapter Assoc. Computational Linguistics (EACL 2026)*, Rabat, Morocco, Mar. 2026, pp. 1430–1440, doi: 10.18653/v1/2026.eacl-long.66. [Online]. Available: https://aclanthology.org/2026.eacl-long.66/ *(Robust over-the-air wake-keyword triggering on Qwen2-Audio. Cited in: §7.9.)*

15.2 A.2 Pre-prints and Technical Reports

*The following sources are pre-prints (arXiv, IACR ePrint, HAL) or technical reports that have **not** undergone formal peer review. They are included because they represent the most current state of research in a rapidly evolving field. Their findings should be interpreted with appropriate caution regarding their pre-review status.*

[3] D. Kong, S. Lin, Z. Xu, Z. Wang, M. Li, Y. Li, et al., "A Survey of LLM-Driven AI Agent Communication: Protocols, Security Risks, and Defense Countermeasures," *arXiv:2506.19676v4*, 2025. [Online]. Available: https://arxiv.org/abs/2506.19676 *(Three-layer architecture; 19 protocols analysed; MCP/A2A vulnerabilities. Cited in: §4.1, §4.6.4, §7.3, §7.4.)*

[7] L. Lim, J. Liu, V. Kalagi, A. El Abbadi, and D. Agrawal, "Hyperion: Private Token Sampling with Homomorphic Encryption," *IACR Cryptology ePrint Archive*, Paper 2025/2318, 2025. [Online]. Available: https://eprint.iacr.org/2025/2318.pdf *(100× latency improvement; 0.14 s for 32 k vocabulary. Cited in: §3.3, §3.4.)*

[8] S. N. Guzelhan, L. Daksha, C. Agulló Domingo, G. Jonatan, J. Kim, J. L. Abellan, et al., "ELLMo: Packing- and Depth-Aware Encrypted Transformer Inference," *IACR Cryptology ePrint Archive*, Paper 2026/198, 2026. [Online]. Available: https://eprint.iacr.org/2026/198 *(Cited in: §3.3, §3.4.)*

[40] Y. Lyu, L. Pan, Y. Xu, and Q. Li, "ZeroSecBench: Fine-grained and Robust Evaluation for Secure Code Generation," submitted to *ICLR 2026* (under review as of Mar. 2026). [Online]. Available: https://openreview.net/forum?id=mTnQkDOh3b *(Best overall Pass@1 ≈ 0.26 for frontier models; 850 vulnerability instances from 150 k GitHub repositories, 12 CWEs, 46 Java components. Status: rejected at ICLR 2026. Cited in: §8.2.1, §8.2.2.)*

[43] A. Tkachenko, D. Suskevic, and B. Adolphi, "Deconstructing Obfuscation: A Four-Dimensional Framework for Evaluating Large Language Models Assembly Code Deobfuscation Capabilities," *arXiv:2505.19887v2*, 2025. [Online]. Available: https://arxiv.org/abs/2505.19887 *(Uses a qualitative 0-5 attacker-knowledge-level scale (not percentages). Tests GPT-4o and Claude 3.7 Sonnet on assembly code deobfuscation across four dimensions. Cited in: §8.3.1.)*

[44] N. Wang, N. Wu, X. Hui, J. Wang, and X. Yuan, "zkUnlearner: A Zero-Knowledge Framework for Verifiable Unlearning with Multi-Granularity and Forgery-Resistance," *arXiv:2509.07290v1*, Sep. 2025. [Online]. Available: https://arxiv.org/abs/2509.07290 *(Zero-knowledge proof framework for verifiable machine unlearning; research prototype only. Cited in: §8.4.3.)*

[45] D. Andreoletti, A. Rudi, E. Carpanzano, F. Lelli, and T. Leidi, "Privacy-Preserving LLM Inference in Practice: A Comparative Survey of Techniques, Trade-Offs, and Deployability," *IACR Cryptology ePrint Archive*, Paper 2026/105, 2026. [Online]. Available: https://eprint.iacr.org/2026/105 *(Comparative survey of TEE, MPC, and FHE for privacy-preserving LLM inference. Cited in: §3.3.)*

[52] T. Cao, B. Lim, Y. Liu, Y. Sui, Y. Li, S. Deng, et al., "VPI-Bench: Visual Prompt Injection Attacks for Computer-Use Agents," in *Proc. Int. Conf. Learning Representations (ICLR 2026)*, 2026. [Online]. Available: https://arxiv.org/abs/2506.02456 *(306-test-case benchmark across 5 platforms; CUAs deceived up to 51%, BUAs up to 100%. Cited in: §7.8.)*

[55] E. Bagdasaryan, T.-Y. Hsieh, B. Nassi, and V. Shmatikov, "Abusing Images and Sounds for Indirect Instruction Injection in Multi-Modal LLMs," *arXiv:2307.10490*, 2023. [Online]. Available:

https://arxiv.org/abs/2307.10490 *(Founding multimodal-injection result: adversarial perturbations in images and audio steer LLaVa/PandaGPT. Cited in: §7.9.)*

[63] N. Nagaraja, L. Zhang, Z. Wang, B. Zhang, and P. Patil, "Image-based Prompt Injection: Hijacking Multimodal LLMs through Visually Embedded Adversarial Instructions," in *Proc. 3rd Int. Conf. Foundation and Large Language Models (FLLM 2025)*, Vienna, Austria, 2025, pp. 916-922. [Online]. Available: https://arxiv.org/abs/2603.03637 *(Up to 64% ASR on GPT-4-Turbo, black-box. Cited in: §7.9.)*

[64] S. Kimura, R. Tanaka, S. Miyawaki, J. Suzuki, and K. Sakaguchi, "Empirical Analysis of Large Vision-Language Models against Goal Hijacking via Visual Prompt Injection," in *Proc. NAACL 2024 Student Research Workshop*, 2024. [Online]. Available: https://arxiv.org/abs/2408.03554 *(15.8% GPT-4V, 6.6% Gemini ASR. Cited in: §7.9.)*

[65] L. Aichberger, A. Paren, G. Li, P. Torr, Y. Gal, and A. Bibi, "MIP against Agent: Malicious Image Patches Hijacking Multimodal OS Agents," in *Proc. Conf. Neural Information Processing Systems (NeurIPS 2025)*, 2025. [Online]. Available: https://arxiv.org/abs/2503.10809 *(Self-propagating adversarial screen imagery. Cited in: §7.9.)*

[67] Z. Ling, P. Hu, X. Gao, X. Ma, M. Zhou, J. Feng, et al., "Sirens' Whisper: Inaudible Near-Ultrasonic Jailbreaks of Speech-Driven LLMs," in *Proc. USENIX Security Symposium (USENIX Security 2026)*, 2026. [Online]. Available: https://arxiv.org/abs/2603.13847 *(Covert single-query inaudible jailbreak delivery; up to 0.94 non-refusal on commercial speech-driven LLMs. Cited in: §7.9.)*

[75] Stanford Institute for Human-Centered Artificial Intelligence (HAI), "Artificial Intelligence Index Report 2026 — Chapter 2: Technical Performance," Stanford HAI, 2026. [Online]. Available: https://aiindex.stanford.edu/report/ *(Open-weight gap reopened in 2025: top closed-weight model leads top open-weight model by 3.3% on the Chatbot Arena as of March 2026, up from 0.5% in August 2024; six of the top 10 Arena models are closed-weight. Downloaded to _aux/_pdf/[Stanford_HAI]_AI_Index_2026_Ch2_Technical.pdf. Cited in: §8.5.8.)*

[76] "The ARC of Progress towards AGI: A Living Survey of Abstraction and Reasoning," arXiv:2603.13372 [cs.AI], 2026 (submitted to ACM Computing Surveys; living survey at https://nimi-ai.com/arc-survey/). [Online]. Available: https://arxiv.org/abs/2603.13372 *(ARC-AGI-1: frontier models exceed 96% — Gemini 3 Deep Think, Opus 4.6 at 93.0%, GPT-5.2 Pro at 90.5%; ARC-AGI-2: Gemini 3 Deep Think 84.6%, Opus 4.6 68.8%; persistent "compositional cliff". Downloaded to _aux/_pdf/2603.13372.pdf. Cited in: §8.5.8.)*

[81] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" in *Proc. Int. Conf. Learning Representations (ICLR 2024)*, 2024. [Online]. Available: https://arxiv.org/abs/2310.06770 *(2,294 real GitHub issues across 12 Python repositories; best model Claude 2 solves 1.96% (4.8% with oracle retrieval); establishes autonomous issue resolution as an evaluation task. Cited in: §7.11.)*

[82] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, et al., "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," in *Proc. Conf. Neural Information Processing Systems (NeurIPS 2024)*, 2024. [Online]. Available: https://arxiv.org/abs/2405.15793 *(Custom agent-computer interface (ACI) lets an LM autonomously search, navigate, edit, and execute to fix GitHub issues; 12.5% pass@1 on SWE-bench, 87.7% on HumanEvalFix; MIT-licensed; also used for offensive cybersecurity. Cited in: §7.11.)*

[83] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, et al., "OpenHands: An Open Platform for AI Software Developers as Generalist Agents," in *Proc. Int. Conf. Learning Representations (ICLR 2025)*, 2025. [Online]. Available: https://arxiv.org/abs/2407.16741 *(Open-source (MIT) platform, formerly OpenDevin and inspired by Devin; generalist agents write code, drive a CLI, and browse the web in sandboxed environments; evaluated across 15 benchmarks incl. SWE-bench and WebArena; 2.1K+ contributions from 188+ contributors. Cited in: §7.11.)*

[84] X. Deng, Y. Zhang, J. Wu, J. Bai, S. Yi, Z. Zou, et al., "Taming OpenClaw: Security Analysis and Mitigation of Autonomous LLM Agent Threats," *arXiv:2603.11619*, 2026. [Online]. Available: https://arxiv.org/abs/2603.11619 *(Five-layer lifecycle security framework (initialization, input, inference, decision, execution) for autonomous LLM agents; tightly-coupled interaction + high-privilege execution "substantially expand the system attack surface"; compound threats (indirect prompt injection, skill supply-chain contamination, memory poisoning, intent drift) defeat point-based defences; calls for holistic security architectures. Cited in: §7.11.)*

[85] M. I. Gorinova, M. Baker, A. Heineike, M. Shaposhnikov, R. Willoughby, and D. Knox, "Position: Coding Benchmarks Are Misaligned with Agentic Software Engineering," *arXiv:2606.17799*, 2026. [Online]. Available: https://arxiv.org/abs/2606.17799 *(Coding benchmarks collapse model, harness, and environment into one end-to-end score, grade against a single reference solution, and give no component-level signal; "a coding agent in practice is not a model: it is a system harness"; surveys 12 benchmarks and finds none measure engineering-process discipline, producing a "lab-to-production gap." Cited in: §7.11.)*

[90] "Silent Egress: Adversarial URL Metadata-Driven Exfiltration in LLM Agents," *arXiv:2602.22450*, 2026. [Online]. Available: https://arxiv.org/abs/2602.22450 *(Metadata-generation side-channels leak runtime context while visible response remains benign; title/metadata generation as silent exfiltration vector. Cited in: §4.6.8.)*

15.3 A.3 Non-Publication Technical Resources

The following are technical resources (websites, project documentation) that are cited as evidence but are not academic publications.

[10] M. S (SnackonAI), "SWE-bench: The Benchmark That Broke Software Engineering AI, Then Got Broken Itself," *snackonai.com*, May 5, 2026. [Online]. Available: https://www.snackonai.com/p/swe-bench-the-benchmark-that-broke-software-engineering-ai-

then-got-broken-itself *(Reports ~43.20% top-agent score on SWE-bench Verified vs ~19.25% on post-cutoff SWE-bench-Live; 8–10% contamination; UTBoost/PatchDiff show ~6–7 absolute-point inflation; up to 345 mis-scored patch assessments; leaderboard governance gap. Retrieved during research; not stored as PDF. Cited in: §7.11.)*

[31] T. Victorino, "The Benchmark Contamination Governance Gap," *victorinollc.com*, 2026. [Online]. Available: https://victorinollc.com/thinking/benchmark-contamination-governance-gap *(Characterises OpenAI's 23 Feb 2026 cessation of SWE-bench Verified score reporting: 59.4% of 138 audited unsolved problems had materially flawed test cases; automated red-teaming showed training contamination across GPT-5.2, Claude Opus 4.5, and Gemini 3 Flash. Retrieved during research; not stored as PDF. Cited in: §7.11, §A.3.)*

[35] Forgepoint Capital, "TIPS #37: When AI Agents Become Insider Threats," *Forgepoint Perspectives*, Mar. 19, 2026. [Online]. Available: https://forgepointcap.com/perspectives/tips-37-when-ai-agents-become-insider-threats/ *(Documents a Feb 2026 incident: prompt injection via a GitHub issue title poisoned an open-source coding agent's CI/CD cache, stole its npm release token, and published a compromised package installing another autonomous agent on ~4,000 developer machines; frames the "semantic von Neumann flaw" and the "lethal trifecta" (private-data access + external communication + untrusted-content exposure); notes autonomy-eroding flags such as --dangerously-skip-permissions / --yolo / -trust-all-tools. Retrieved during research; not stored as PDF. Cited in: §7.11.)*

[36] Georgia Tech, Purdue University, and Synkhronix, "TEE.fail: Breaking Trusted Execution Environments via DDR5 Memory Bus Interposition," Oct. 2025. [Online]. Available: http://tee.fail/ *(Physical attack against Intel SGX, TDX, AMD SEV-SNP using <$1,000 hardware; DDR5 memory bus interposition extracts encryption keys and enclave secrets. Cited in: §3.4, §8.4.3, §8.5.6.)*

[57] Cloud Security Alliance, "Computer-Use Agent Safety Blind Spots," *CSA Research Note*, Apr. 15, 2026. [Online]. Available: https://labs.cloudsecurityalliance.org/research/csa-research-note-computer-use-agent-safety-blindspots-20260/ *(Adversarial pop-ups 86% mean ASR; screenshot-based agents treat rendered screen as trusted instruction context. Cited in: §7.8.)*

[58] HiddenLayer, "Indirect Prompt Injection of Claude Computer Use," *HiddenLayer Research Blog*, Oct. 24, 2024. [Online]. Available: https://www.hiddenlayer.com/research/indirect-prompt-injection-of-claude-computer-use *(Demonstrates Claude Computer Use executing rm -rf from an obfuscated payload in a local PDF. Cited in: §7.8.)*

[59] OpenAI, "Introducing Operator," *OpenAI Blog*, Jan. 23, 2025. [Online]. Available: https://openai.com/index/introducing-operator/ *(Computer-Using Agent (CUA) model; takeover mode, user confirmations, monitor model. Cited in: §7.8.)*

[60] OpenAI, "Continuously Hardening ChatGPT Atlas Against Prompt Injection Attacks," *OpenAI Blog*, 2025. [Online]. Available: https://openai.com/index/hardening-atlas-against-prompt-injection/ *(Browser-based prompt injection "may never be fully solved." Cited in: §7.8.)*

[61] Google Security Research, "OpenAI Operator Security Advisories," *github.com/google/security-research*, 2025. [Online]. Available: https://github.com/google/security-research/security/advisories/GHSA-25j5-vvch-9rf3 *(OAuth code and credential exfiltration via TOCTOU. Cited in: §7.8.)*

[62] Anthropic, "Computer Use Tool — Claude Documentation," Anthropic, 2025. [Online]. Available: https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/computer-use-tool *(Recommends dedicated VM/container, credential isolation, URL allowlists. Cited in: §7.8.)*

[68] Cursor, "Ignore File," Cursor Docs, 2026. [Online]. Available: https://docs.cursor.com/en/context/ignore-files (canonical: https://cursor.com/docs/reference/ignore-file) *(.cursorignore blocks semantic search, Tab, Agent inline-edit, and @-mentions only; terminal and MCP server tools used by Agent cannot block access to code governed by .cursorignore; "complete protection isn't guaranteed due to LLM unpredictability." Archived: _aux/_pdf/[Cursor] ignore-files.html. Cited in: §7.10, §11.1, §A.3.)*

[69] Cursor, "Security," Cursor (Anysphere, Inc.), last updated Apr. 24, 2026. [Online]. Available: https://cursor.com/en-US/security *(All app requests route through Cursor backend domains for API, indexing, update, and marketplace functionality; Privacy Mode, when enabled, prevents training on user data and adds provider-side contractual controls. Archived: _aux/_pdf/[Cursor] security.html. Cited in: §7.10, §11.1.)*

[70] Anthropic, "Best practices for Claude Code," Anthropic Engineering, 2026. [Online]. Available: https://www.anthropic.com/engineering/claude-code-best-practices *(Describes Claude Code as an agentic environment that "can read your files, run commands, make changes, and autonomously work through problems while you watch, redirect, or step away entirely." Archived: _aux/_pdf/[Anthropic] claude-code-best-practices.html. Cited in: §7.10, §11.1.)*

[71] OpenCode (SST / Anomaly), "The open source AI coding agent," 2026. [Online]. Available: https://opencode.ai/ (source: https://github.com/sst/opencode) *(MIT-licensed; ~176k GitHub stars; v1.17.8 (2026-06-17); 75+ LLM providers incl. local; "OpenCode does not store any of your code or context data." Archived: _aux/_pdf/[Tool] OpenCode.html. Cited in: §11.1.1, §A.3.)*

[72] Cline Bot Inc., "Cline — AI Coding, Open Source and Uncompromised," 2026. [Online]. Available: https://cline.bot/ (source: https://github.com/cline/cline) *(Apache 2.0; ~63.7k stars; 250+ contributors; actively maintained through 2026; BYO key/weights; hosted ToS applies only to the optional Cline API provider. Archived: _aux/_pdf/[Tool] Cline.html. Cited in: §11.1.2, §A.3.)*

[73] OpenAI, "Using Codex with your ChatGPT plan," OpenAI Help Center, 2026. [Online]. Available: https://help.openai.com/en/articles/11369540-using-codex-with-your-chatgpt-plan *(Proprietary; cloud containers + local OS sandbox; network off by default; training off by default for Business/Enterprise/Edu; in-app browser/CDP risk surfaces. Archived: _aux/_pdf/[Tool] Codex.html. Cited in: §11.1.3.)*

[74] Cognition, "Devin Desktop" (formerly Windsurf), 2026. [Online]. Available: https://windsurf.com/windsurf *(Proprietary; Codeium → Windsurf → Cognition acquisition → Devin Desktop; multi-agent command center over ACP; cloud handoff. Archived: _aux/_pdf/[Tool] Windsurf.html. Cited in: §11.1.3.)*

[86] Cognition, "Introducing Devin, the first AI software engineer," *Cognition Blog*, Mar. 12, 2024. [Online]. Available: https://cognition.ai/blog/introducing-devin *(Devin operates in a sandboxed shell+editor+browser environment, making "thousands of decisions" over long-horizon tasks; reports 13.86% unassisted resolution on a 25% subset of SWE-bench vs 1.96% prior state-of-the-art. Retrieved during research; not stored as PDF. Cited in: §7.11.)*

[87] OpenHands (All-Hands AI), "OpenHands — The Open Platform for Cloud Coding Agents," 2026. [Online]. Available: https://github.com/All-Hands-AI/OpenHands *(MIT-licensed, self-hosted "developer control center"; orchestrates OpenHands/Claude Code/Codex/Gemini over ACP on local, remote, or private-cloud backends; sandboxed execution; agents run locally by default. Retrieved during research; not stored as PDF. Cited in: §7.11.)*

[88] SWE-bench, "SWE-bench — Can Language Models Resolve Real-World GitHub Issues?" *swebench.com*, 2024. [Online]. Available: https://www.swebench.com/original.html *(Project/leaderboard site for [81]; documents the SWE-agent 12.47% result and the SWE-bench Verified split used by subsequent systems. Retrieved during research; not stored as PDF. Cited in: §7.11.)*

[89] Cloud Security Alliance, "Agents in the Wire: AI Agents as Enterprise Insider Threats (Autonomous Compromise v1)," *CSA Labs Research*, Mar. 25, 2026. [Online]. Available: https://labs.cloudsecurityalliance.org/research/ai-agent-insider-threat-autonomous-compromise-v1-csa-styled/ *(2025 CSA Agentic Identity Survey: 82% of organisations not highly confident in IAM for agent identities, only 17% enforce runtime access controls consistently; documents zero-click Copilot exfiltration (EchoLeak), Trail of Bits argument-injection RCE chains (CVE-2025-54795 in Claude Code, GHSA in Cursor, Amazon Q), and ReversingLabs on agents with repository write access introducing backdoors at scale. Retrieved during research; not stored as PDF. Cited in: §7.11.)*

[91] KiloCode (Kilo-Org), "The open source AI coding agent," 2026. [Online]. Available: https://github.com/Kilo-Org/kilocode *(MIT-licensed fork of OpenCode; hidden title agent with undocumented model-selection fallback chain. Source files: packages/opencode/src/agent/agent.ts (title agent definition, lines 262–277), packages/opencode/src/session/prompt.ts (ensureTitle model selection, lines 369–372),*

packages/opencode/src/provider/provider.ts (getSmallModel priority list). Cited in: §4.6.8, §11.1.1.)

[92] KiloCode, "Use separate taskID for title generation to prevent model leak," Pull Request #6733, 2026. [Online]. Available: https://github.com/Kilo-Org/kilocode/pull/6733 *(Maintainer acknowledgement of title-generation model-leak class; commit b1ab641 "add small model for title generation"; related issue #6552 "LLM.stream small-model requests leak into subsequent agent session." Cited in: §4.6.8.)*

15.4 B. Industry Standards and Frameworks

[11] OWASP Foundation, "OWASP Top 10 for LLM Applications 2025," OWASP, 2025. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/ *(LLM01–LLM10 vulnerability taxonomy. Cited in: §4.6.8, §5.3, §5.5, §10.5.)*

[12] OWASP Foundation, "OWASP Top 10 for Agentic Applications 2026," OWASP, 2026. [Online]. Available: https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/ *(ASI01–ASI10 vulnerability taxonomy. Cited in: §4.6.8, §7.2, §10.5.)*

[13] OWASP GenAI Security Project, "GenAI Security Project Solutions Reference Guide Q2/Q3 2025," OWASP, 2025. [Online]. Available: https://genai.owasp.org/ *(80+ vendors mapped across the AI security lifecycle. Cited in: §13.5.)*

[14] OWASP GenAI Security Project, "GenAI COMPASS RunBook 1.0," OWASP, 2025. [Online]. Available: https://genai.owasp.org/ *(OODA loop methodology; 5-point scoring system.)*

[15] OWASP GenAI Security Project, "A Practical Guide for Securely Using Third-Party MCP Servers," ver. 1.0, OWASP, Nov. 4, 2025. [Online]. Available: https://genai.owasp.org/resource/cheatsheet-a-practical-guide-for-securely-using-third-party-mcp-servers-1-0/ *(MCP transport modes, tool poisoning, rug pull attacks, OAuth 2.1 gaps, 5-step enterprise governance workflow. License: CC BY-SA 4.0. Cited in: §4.6.2, §4.6.4, §4.6.5, §7.10.)*

[16] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, NIST, Gaithersburg, MD, USA, Jan. 2023. [Online]. Available: https://doi.org/10.6028/NIST.AI.100-1 *(GOVERN/MAP/MEASURE/MANAGE functions; Seven Trustworthiness Characteristics. Cited in: §10.5.)*

[17] National Cyber Security Centre (NCSC) and Cybersecurity and Infrastructure Security Agency (CISA), "Guidelines for Secure AI System Development," NCSC/CISA, 2023. [Online]. Available: https://www.ncsc.gov.uk/collection/guidelines-secure-ai-system-development *(Four lifecycle areas; endorsed by 20+ international agencies. Cited in: §10.5.)*

[18] OWASP Foundation, "State of Agentic AI Security and Governance v1.0," OWASP, 2025. [Online]. Available: https://genai.owasp.org/resource/state-of-agentic-ai-security-and-governance/ *(Agent taxonomy; framework comparison.)*

15.5 C. Vulnerability Advisories (2025–2026)

[19] Orca Security, "Exploiting GitHub Copilot for a Repository Takeover," *Orca Security Blog*, Feb. 16, 2026. [Online]. Available: https://orca.security/resources/blog/roguepilot-github-copilot-vulnerability/ *(Cited in: §5.2, §5.6, §11.1, §A.3.)*

[20a] NIST National Vulnerability Database, "CVE-2026-2256," NIST, Gaithersburg, MD, USA, Mar. 2, 2026. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2026-2256 *(CVSS 3.1 = 6.5 Medium (CISA-ADP); CWE-77 Command Injection; affected product: ModelScope ms-agent <= v1.6.0rc1. Cited in: §2.4.)*

[20] I. Yochpaz, "CVE-2026-2256: From AI Prompt to Full System Compromise," *Medium*, Feb. 25, 2026. [Online]. Available: https://medium.com/@itamar.yochpaz/cve-2026-2256-from-ai-prompt-to-full-system-compromise *(Cited in: §2.4, §5.2.)*

[21] Check Point Research, "Caught in the Hook: RCE and API Token Exfiltration Through Claude Code Project Files (CVE-2025-59536, CVE-2026-21852)," *Check Point Research Blog*, Feb. 25, 2026. [Online]. Available: https://research.checkpoint.com/2026/rce-and-api-token-exfiltration-through-claude-code-project-files-cve-2025-59536/ *(Cited in: §5.2.)*

[22] Palo Alto Networks Unit 42, "Fooling AI Agents: Web-Based Indirect Prompt Injection Observed in the Wild," *Unit 42 Threat Research*, Mar. 3, 2026. [Online]. Available: https://unit42.paloaltonetworks.com/ai-agent-prompt-injection/ *(Cited in: §5.2, §5.6, §7.8.)*

[23] Embrace The Red, "Claude Code: Data Exfiltration with DNS (CVE-2025-55284)," *Embrace The Red Blog*, Aug. 2025. [Online]. Available: https://embracethered.com/blog/posts/2025/claude-code-exfiltration-via-dns-requests/ *(Cited in: §5.2, §5.6.)*

15.6 D. Industry Reports and Surveys

[24] Arctic Wolf Networks, "Arctic Wolf 2025 Trends Report," Arctic Wolf Networks, Eden Prairie, MN, USA, May 2025. [Online]. Available: https://arcticwolf.com/resources/press-releases/arctic-wolf-2025-trends-report/ *(AI ranked #1 security concern (29%) vs. ransomware (21%); 1,200+ respondents, 15 countries.)*

[25] World Economic Forum, "Global Cybersecurity Outlook 2026," WEF, Geneva, Switzerland, Jan. 2026. [Online]. Available: https://www.weforum.org/publications/global-cybersecurity-

outlook-2026/ *(87% of leaders identify AI vulnerabilities as fastest-growing cyber risk; 804 participants, 92 countries. Cited in: §11.4.)*

[26] Menlo Ventures, "2025: The State of Generative AI in the Enterprise," Menlo Ventures, Menlo Park, CA, USA, Dec. 2025. [Online]. Available: https://menlovc.com/perspective/2025-the-state-of-generative-ai-in-the-enterprise/ *(Shadow AI: 27% of ChatGPT Plus usage is work-related, indicating significant unsanctioned enterprise use through personal accounts.)*

[27] LocalAIMaster, "Local vs Cloud LLM Deployment: Cost Analysis 2025," *localaimaster.com*, 2025. [Online]. Available: https://localaimaster.com/blog/local-vs-cloud-llm-deployment-strategies (accessed 2026-06-23) *(214-enterprise study; hybrid architecture: 60% cost reduction; 40% adoption rate. Cited in: §8.5.8, §8.6, §11.2, §A.3.)*

[28] Dark Reading, "2026: The Year Agentic AI Becomes the Attack-Surface Poster Child," *Dark Reading*, Jan. 30, 2026. [Online]. Available: https://www.darkreading.com/threat-intelligence/2026-agentic-ai-attack-surface-poster-child *(Cited in: §4.6.2.)*

[29] Tech Law Blog, "EU AI Act Timeline Update," *TechLaw.ie*, Mar. 6, 2026. [Online]. Available: https://www.techlaw.ie/2026/03/articles/artificial-intelligence/eu-ai-act-timeline-update/ *(Cited in: §10.1.)*

[30] Help Net Security, "Enterprises Are Racing to Secure Agentic AI Deployments," *Help Net Security*, Feb. 23, 2026. [Online]. Available: https://www.helpnetsecurity.com/2026/02/23/ai-agent-security-risks-enterprise/

[32] Gartner, "Shadow AI Statistics 2026: Unauthorized AI Tool Usage in Enterprise," Gartner, Stamford, CT, USA, 2026. *(68% of employees use unauthorised AI tools without IT approval. Cited via [33]; Gartner report not publicly accessible — data point attributed to [33]. Cited in: §4.6.3.)*

[33] E. Chan, "Top 50 Shadow AI Statistics 2026: Real Data on Hidden AI Use," *Second Talent*, Mar. 4, 2026. [Online]. Available: https://www.secondtalent.com/resources/shadow-ai-stats/ *(80%+ of enterprises have no visibility into AI tool usage; 12% of companies can detect all shadow AI usage. Cited in: §4.6.3.)*

[34] L. Cardiet, "Shadow AI Explained: The Unsanctioned AI Risk Hiding in Every Enterprise," *Vectra AI*, May 6, 2026. [Online]. Available: https://www.vectra.ai/topics/shadow-ai *(Average additional breach cost from Shadow AI: $670,000; annual insider risk cost from AI negligence: $10.3 M; ~50% of employees continue AI use after a ban. Cited in: §4.6.3.)*

[46] Cisco, "Cisco AI Readiness Index 2025," Cisco Systems, San Jose, CA, USA, 2025. [Online]. Available: https://www.cisco.com/c/m/en_us/solutions/ai/readiness-index/realizing-the-value-of-ai.html (accessed 2026-06-23) *(3rd edition; 83% of orgs plan AI agent deployment; agentic systems "take action by navigating tasks, workflows, and even business decisions autonomously." Cited in: §4.5, §11.6.)*

[47] Basebox.ai, "Practical Insights and Tooling for On-Premise LLMs," *Basebox.ai*, 2025. [Online]. Available: https://basebox.ai/ *(On-premise LLM deployment cost/benefit analysis; infrastructure requirements. Cited in: §8.5.3.)*

[48] Docker, "MCP Horror Stories, Part 3: The Data Heist," *Docker Blog*, 2025. [Online]. Available: https://www.docker.com/blog/ *(Analysis of GitHub MCP Data Heist incident; "Always Allow" configuration risks. Cited in: §16.1 LL-1, §16.1 LL-2.)*

[49] Invariant Labs, "GitHub MCP Exploited: Accessing private repositories via MCP," *Invariant Labs*, May 26, 2025. [Online]. Available: https://invariantlabs.ai/blog/github-mcp-exploited *(Zero-click prompt injection via malicious GitHub issue; autonomous data exfiltration through MCP server. Cited in: §16.1 LL-1, §16.1 LL-2, §7.3.)*

[77] Artificial Analysis, "Recent Open Weights Model Launches," artificialanalysis.ai, Apr. 2026. [Online]. Available: https://artificialanalysis.ai/articles/recent-open-weights-model-launches *(Top open-weight models within 3–6 points of the leading proprietary model on the Artificial Analysis Intelligence Index, but the gap is wide on the hardest reasoning/agentic coding: HLE ~34–36% open vs ~44–45% closed; Terminal-Bench Hard ~43–46% open vs ~61% closed; CritPt (research-level physics) ~4–12% open vs ~27% closed. Cited in: §8.5.8.)*

[78] S. (Gen α AI), "AI Models 2026: The Mid-Year Frontier and Open-Weight Map," genalphai.com, Jun. 2026. [Online]. Available: https://genalphai.com/the-2026-ai-model-landscape/ *(Open-weight cluster within ~3 points of the frontier on standard benchmarks — AIME, MMLU-Pro, GPQA, SWE-bench Verified — but trails by 10–25 points on agentic/hardest-reasoning benchmarks; large gap on ARC-AGI-2/3, Terminal-Bench 2.0, Humanity's Last Exam, and SWE-bench Pro. Cited in: §8.5.8.)*

[79] Vals AI, "SWE-bench Verified," vals.ai, 2026. [Online]. Available: https://vals.ai/benchmarks/swebench *(Independent bash-tool-only harness; closed-source models consistently outperform open-source models on SWE-bench Verified, with the clearest performance differences on tasks taking 15 minutes to 1 hour. Cited in: §8.5.8.)*

[80] H. Ihle, "open_closed_gap — Measuring the Open-vs-Closed Model Delay," GitHub repository, 2026 (analysis over Epoch AI Benchmarking Hub data). [Online]. Available: https://github.com/htihle/open_closed_gap *(Contamination-resistant/private benchmarks show roughly 1.7–1.9× the open-vs-closed gap of public benchmarks. Secondary analysis; cited with caution. Cited in: §8.5.8.)*

15.7 E. Regulatory and Official Sources

[50] European Parliament, "Artificial Intelligence Act: delayed application, ban on nudifier apps," Press Release, Ref. 20260323IPR38829, Mar. 26, 2026. [Online]. Available: https://www.europarl.europa.eu/news/en/press-room/20260323IPR38829/artificial-intelligence-act-delayed-application-ban-on-nudifier-apps *(Parliament negotiating position adopted 26 March 2026 by 569–45–23; proposes delaying high-risk AI application to 2 Dec 2027*

(listed systems), 2 Aug 2028 (sectoral), and watermarking to 2 Nov 2026; trilogue with Council pending. Cited in: §10.1.)

[51] European Commission, "The General-Purpose AI Code of Practice," Shaping Europe's Digital Future, Jul. 10, 2025. [Online]. Available: https://digital-strategy.ec.europa.eu/en/policies/contents-code-gpai *(Voluntary AI Act compliance tool for GPAI model providers; three chapters: Transparency, Copyright, and Safety & Security; endorsed by the Commission and the AI Board as an adequate compliance tool under AI Act Articles 53 and 55. Cited in: §10.1.1.)*

---

16 Lessons Learned and Practical How-To Guide

About this section: This section distils practical lessons learned during the InnovAIte project's own experience integrating AI coding assistants into enterprise software development workflows. Unlike the preceding sections — which synthesise external research and industry standards — the content here reflects first-hand observations, configuration decisions, and operational patterns encountered by the project team. Where a lesson is grounded in an external incident or standard, the source is cited. Where it reflects internal project experience, it is labelled *[Project experience]*.

16.1 Lessons Learned: What We Discovered in Practice

16.1.1 LL-1: The "Always Allow" Trap — How Convenience Becomes a Vulnerability

What happened: During routine use of agentic coding assistants (RooCode, Cursor), developers are repeatedly prompted to confirm individual tool calls — file reads, shell commands, web fetches. The natural response to this friction is to click **"Always Allow"** for a given tool category. This is the single most dangerous configuration decision a developer can make.

Why it matters: The GitHub MCP Data Heist (Invariant Labs, May 2025) [49] demonstrated this precisely: the attack succeeded because the victim had enabled an "Always Allow" confirmation policy. Once the AI agent read a malicious GitHub issue containing hidden prompt injection instructions, it autonomously accessed private repositories and exfiltrated salary data — all without any further developer interaction. The developer never saw the attack happen [48].

Lesson: Agent tool confirmations are not UX friction — they are the last human-in-the-loop checkpoint before autonomous action. Disabling them removes the only control that can catch a prompt injection attack mid-execution.

Rule of thumb: Never use "Always Allow" for tool categories that involve: (a) cross-repository access, (b) shell command execution, (c) external network requests, or (d) file writes outside the current project directory.

Mitigation cross-reference: §5.3 (EchoLeak mitigation matrix — Human-in-the-Loop is the only control that prevents zero-click exfiltration); §9.9 (Guardian Agent Pattern).

16.1.2 LL-2: Broad Personal Access Tokens Are Attack Multipliers

What happened: The standard documentation for most GitHub MCP integrations instructs developers to configure a single Personal Access Token (PAT) with broad repository access. This is the path of least resistance — and the path of maximum risk.

Why it matters: A single broad PAT grants the AI agent access to *every* repository the developer can access — public projects, private company repositories, personal code. When the agent is prompt-injected via a malicious public repository issue, it can immediately pivot to private repositories using the same credential. The attack requires no additional privilege escalation because the token already has it [48].

Lesson: The scope of a credential defines the blast radius of a compromise. A broad PAT turns a prompt injection in a public repository into a full private repository breach.

Rule of thumb: Every AI tool integration should use the minimum credential scope required for the specific task. For GitHub: use fine-grained PATs scoped to specific repositories, not classic PATs with broad access. Prefer OAuth with repository-specific scopes over PATs entirely.

Mitigation cross-reference: §4.6.5 (OAuth scope over-provisioning); §7.3 (MCP authentication gaps); §13.6 (MCP governance workflow).

16.1.3 LL-3: The Context Window Is Larger Than You Think

What happened: Developers typically believe they are submitting only the code they are actively editing to the AI model. In practice, agentic coding assistants aggregate context from multiple sources simultaneously: the active file, other open files in the IDE, terminal output, file names, import statements, and cross-file references. This aggregation is largely invisible.

Why it matters: A developer asking for help with a utility function may inadvertently expose the entire workspace context — including files containing API keys, database connection strings, or proprietary business logic in adjacent files. The AAAI-25 research confirmed that the Filename Proxy Attack (72.5% ASR on GitHub Copilot) exploits exactly this: the file name itself is

part of the context, and a sensitive file name can cause the model to generate content aligned with that sensitivity. [§6.3.1]

Lesson: The context window is not what you see in the chat — it is everything the agent can read. Sensitive files that are merely *open* in the IDE may be transmitted.

Rule of thumb: Before starting an AI-assisted coding session on a sensitive module, close all unrelated files. Use .rooignore (or equivalent) to explicitly exclude sensitive directories from agent context aggregation. Treat the context window as a data transmission boundary, not a chat window.

Mitigation cross-reference: §4.2 (implicit context gathering); §6.2 (LCCT workflow vulnerability points); LL-4 (.rooignore how-to below).

16.1.4 LL-4: .rooignore Is the First Line of Defence — Use It

What happened: During project setup, the team discovered that RooCode (and similar agentic IDEs) respect a .rooignore file analogous to .gitignore. Without explicit configuration, the agent has access to the entire workspace directory tree, including configuration files, credential stores, and sensitive modules.

Why it matters: The .rooignore file allows developers to explicitly exclude files and directories from the agent's file system access. This is a direct implementation of the Prevention/Isolation paradigm (§8.2) at the IDE level — the simplest and lowest-cost control available. [Project experience; RooCode documentation]

Lesson: Every repository used with an agentic coding assistant should have a .rooignore file configured before the first AI session. This is not optional hygiene — it is a mandatory security control.

Rule of thumb: At minimum, .rooignore should exclude: .env files and directories, credential stores (*.pem, *.key, *.p12), configuration files with connection strings, directories containing cryptographic implementations, and any directory classified as Confidential or Restricted.

Mitigation cross-reference: §8.2 (Prevention/Isolation paradigm); §9.4 (air-gapped mandates for high-sensitivity workloads).

16.1.5 LL-5: Shadow AI Starts with the Developer Next to You

What happened: During the project, it became apparent that developers under deadline pressure routinely used personal AI accounts (personal ChatGPT, personal GitHub Copilot

subscriptions) to process code snippets from the project codebase — outside all enterprise controls, ZDR agreements, and audit logging.

Why it matters: Shadow AI is not a theoretical risk — it is the default behaviour when sanctioned tools are unavailable, slow, or restricted. The Gartner finding that 68% of employees use unauthorised AI tools [§4.6.3] is consistent with project observations. Each personal account submission bypasses every enterprise control described in this report.

Lesson: Prohibition alone does not work — approximately 50% of employees continue AI use after a ban [§4.6.3]. The effective response is to make the sanctioned tool *better* than the unsanctioned alternative: faster, more capable, and easier to use. A local LLM proxy (LiteLLM) that routes to both local models and governed cloud endpoints, accessible from the developer's IDE with a single configuration, removes the incentive to use personal accounts.

Rule of thumb: Measure Shadow AI before trying to eliminate it. Deploy network monitoring to identify traffic to api.openai.com, api.anthropic.com, generativelanguage.googleapis.com from developer endpoints. Distinguish sanctioned (enterprise account) from unsanctioned (personal account) traffic by API key prefix or certificate. Address the root cause (tool friction) before the symptom (policy violation).

Mitigation cross-reference: §4.6.3 (Shadow AI as perimeter leakage vector); §11.2 (enterprise adoption landscape); §13.4 (P0 action: Shadow AI audit).

16.1.6 LL-6: MCP Server Installation Is a Supply Chain Event — Treat It as One

What happened: The MCP ecosystem grew from a developer convenience feature to a critical security boundary within months of its introduction. The team observed that developers installed MCP servers from public registries (npm, PyPI) without any vetting process — the same behaviour that led to the Postmark MCP impersonation (Feb 2026) and the MCP Filesystem Server unauthenticated Bash shell (port 4444).

Why it matters: Installing an MCP server is equivalent to installing a privileged plugin that can read your files, execute commands, and transmit data to external endpoints. The tool description itself can contain hidden prompt injection instructions that the LLM will execute. [§4.6.4; §7.3]

Lesson: Every MCP server installation must be treated as a supply chain event: source verification, version pinning, hash verification, and periodic re-validation. The OWASP Practical Guide for Securely Using Third-Party MCP Servers (October 2025) provides the definitive checklist. [§13.6]

Rule of thumb: Before installing any MCP server: (1) verify the publisher identity matches the claimed organisation, (2) read the full tool description for suspicious instructions or markup, (3) check the npm/PyPI package for recent ownership changes, (4) pin the version and record the hash, (5) test in an isolated environment before production use.

Mitigation cross-reference: §7.3 (MCP vulnerability categories); §7.4 (AI-mediated supply chain attacks); §13.6 (MCP governance workflow).

16.1.7 LL-7: Prompt Caching and Identity Leakage — An Underappreciated Risk

What happened: During investigation of the LiteLLM proxy configuration (used to route requests to Azure OpenAI), the team identified a subtle risk: prompt caching at the proxy layer can cause context from one developer's session to bleed into another developer's session if cache keys are not properly scoped to user identity.

Why it matters: This maps directly to OWASP ASI03 (Identity and Privilege Abuse) and ASI06 (Memory and Context Poisoning). If the LiteLLM proxy caches prompt prefixes without binding the cache key to the authenticated user identity, a developer may receive completions that include context from a colleague's session — potentially including code from a project they do not have access to. [Project experience; §7.2 ASI03, ASI06]

Lesson: Prompt caching is a performance optimisation that introduces a data isolation risk. Cache keys must include the authenticated user identity (not just the prompt hash) to prevent cross-user context bleed.

Rule of thumb: When configuring LiteLLM or any LLM proxy with prompt caching: (1) ensure cache keys include the user identity token, (2) set appropriate TTL (time-to-live) on cached entries, (3) disable caching entirely for sessions involving Confidential or Restricted code, (4) audit cache hit/miss logs for anomalous cross-user patterns.

Mitigation cross-reference: §7.2 (ASI03, ASI06); §9.5 (RAG security boundaries and identity propagation); §9.8 (audit logging requirements).

16.1.8 LL-8: The "Rug Pull" Risk in Long-Running Projects

What happened: The team observed that MCP servers and AI tool plugins that were vetted and approved at project inception may be silently updated by their publishers months later. A server that was safe in October 2025 may have been compromised by February 2026 — without any new installation event triggering re-review.

Why it matters: The Rug Pull attack (§4.6.4) exploits exactly this: a previously legitimate server is updated to include malicious behaviour. Because the server was previously approved, it bypasses re-validation controls. The Framelink Figma MCP RCE (Feb 2026) demonstrated that even widely-used, apparently legitimate MCP servers can introduce RCE vulnerabilities through updates.

Lesson: MCP server approval is not a one-time event — it is an ongoing monitoring obligation. Version pinning is the primary technical control; periodic re-validation is the governance control.

Rule of thumb: Pin all MCP server versions in your configuration. Subscribe to security advisories for every MCP server in use. Implement automated hash verification on startup — if the hash of the installed server does not match the approved hash, block startup and alert the security team.

Mitigation cross-reference: §4.6.4 (Rug Pull attack); §7.3 (MCP security controls — Monitor Tool Integrity); §13.6 (MCP governance workflow — Re-validation step).

16.1.9 LL-9: Inline Sensitive-Data Detection as a LiteLLM Callback — Qwen3Guard

What happened: After deploying the LiteLLM proxy as the enterprise AI gateway, the project team identified a gap: while all prompts were now routed through a single controlled endpoint, there was no automated mechanism to detect when a developer was inadvertently submitting sensitive data — credentials, PII, proprietary business logic — in a prompt. Human review of every prompt was not operationally feasible.

What we did: The team deployed **Qwen3Guard**, a specialised sensitive-data detection model, as a **LiteLLM callback**. Every prompt passing through the LiteLLM proxy is evaluated by Qwen3Guard before being forwarded to the target LLM. Prompts classified as containing sensitive data are flagged in the audit log; optionally, they can be blocked or routed to a local-only model. [Project experience]

Why it matters: This implements the "AI Firewall" pattern described in §9.9 (Guardian Agent Pattern) at the proxy layer — without requiring any changes to developer tooling or workflow. The detection operates transparently: developers continue using their IDE as normal, while the proxy enforces the data classification policy. This directly addresses the gap identified in §8.4 (Governed Secure Channel Architectures): ZDR agreements are contractual controls, not technical ones. Qwen3Guard provides a technical enforcement layer.

Lesson: A sensitive-data detection callback at the LLM proxy layer is the most operationally efficient control available: it covers all AI tools simultaneously (IDE assistants, chat interfaces,

API consumers), requires no per-tool configuration, and produces an audit trail that satisfies compliance requirements.

Rule of thumb: Deploy sensitive-data detection as a proxy-layer callback, not as a client-side plugin. Client-side controls can be bypassed by switching tools; proxy-layer controls cannot. Use a model specifically fine-tuned for sensitive-data classification (not a general-purpose LLM) to minimise latency and false-positive rates.

Mitigation cross-reference: §8.4 (Governed Secure Channel — technical vs. contractual controls); §9.9 (Guardian Agent Pattern); §13.4 (P0 action: audit logging); §11.4 (AI gateway deployment patterns).

16.1.10 LL-10: LiteLLM as a Centralised MCP Gateway — Curation, Logging, and Firewalling

What happened: As MCP server adoption grew within the project, the team faced a governance problem: each developer could independently install and configure MCP servers in their IDE, creating an uncontrolled proliferation of tool integrations with no central visibility, no audit trail, and no consistent vetting process. The MCP server landscape was effectively a shadow IT problem within the AI tooling stack.

What we did: The team leveraged **LiteLLM's central MCP functionality** to route approved MCP servers through the LiteLLM proxy. Instead of each developer connecting directly to MCP servers, all MCP traffic is routed through the LiteLLM endpoint using the developer's LiteLLM API key. This architecture provides three critical capabilities: [Project experience]

1. **MCP server curation:** Only MCP servers registered in the LiteLLM configuration are accessible. Unapproved servers are unreachable by design — developers cannot connect to an MCP server that is not on the approved list.
2. **Centralised logging:** All MCP tool calls (tool name, parameters, response) are logged through the LiteLLM audit trail, providing the visibility required for incident response and compliance.
3. **Protocol-level firewalling:** Network-level firewall rules block all direct MCP protocol traffic (typically WebSocket or SSE on port 443) except to the LiteLLM endpoint. This means even if a developer attempts to connect to an unapproved MCP server directly, the connection is blocked at the network layer.

Why it matters: This architecture transforms MCP governance from a policy problem (asking developers to self-enforce) into a technical enforcement problem (making non-compliant behaviour impossible). It directly addresses LL-6 (MCP supply chain risk) and LL-8 (Rug Pull risk)

by centralising the point of control: when a new MCP server version is released, the security team reviews and approves it once in the LiteLLM configuration, and the update is immediately effective for all developers.

Lesson: Centralising MCP server access through the LLM proxy converts a distributed governance problem into a single configuration management problem. The combination of API-key-based access control, centralised logging, and network-level firewalling provides defence-in-depth for MCP tool usage.

Rule of thumb: Treat the LiteLLM proxy as the single authorised MCP gateway for the organisation. Block all direct MCP connections at the network perimeter. Maintain the approved MCP server list in the LiteLLM configuration as a version-controlled artefact, subject to the same change management process as production infrastructure.

Mitigation cross-reference: §7.3 (MCP vulnerability categories — authentication gaps); §13.6 (MCP governance workflow); §11.4 (enterprise AI gateway patterns); LL-6 (MCP supply chain); LL-8 (Rug Pull monitoring).

16.1.11 LL-11: Hidden Default Model Selection in Built-In Agents

What happened: During traffic analysis of KiloCode API calls, the team observed the user's initial prompt being sent to google/claude-haiku-4-5 — a model never explicitly configured by the user. Investigation revealed a hidden title agent that fires on the first turn of every session [91]. When neither agent.title.model nor small_model is configured, model selection falls through to a hardcoded priority list, silently routing sensitive prompts to unexpected cloud models. A parallel hidden summary agent was identified with the same model-routing risk.

Why it matters: A developer who has carefully configured local inference or ZDR agreements for their primary agent may still have their opening prompt (including attached file contents) egress to a different cloud provider. The title-generation prompt explicitly instructs the model to preserve technical terms and filenames, forwarding sensitive material verbatim. This defeats the local-model/ZDR guarantee — the central recommendation of this report for high-sensitivity workloads (§8.5, §9.3–9.4).

Lesson: Auxiliary agent calls (title generation, conversation summarisation, commit message generation, prompt enhancement) are first-class egress and must be governed as such. "I configured my model" is insufficient — you must verify *every* model call the harness makes, including hidden internal agents.

Rule of thumb: Audit every LLM.stream({ small: true }) call site and every hidden-agent invocation in your AI coding assistant. Either explicitly bind all auxiliary agents to your governed model, disable them, or route all traffic through a DLP proxy that catches unconfigured-provider egress. See §4.6.8 for the full technical analysis and §9.2 for DLP proxy configuration.

16.2 How-To: Pre-Session Checklist Before Using an AI Coding Assistant

Use this checklist before starting any AI-assisted coding session involving internal or sensitive code.

Step 1 — Classify the session

Question	If YES	If NO
Does the session involve Security-Critical Modules (SCMs)?	Use local LLM only; apply Prevention	Cloud LLM acceptable with ZDR
Does the session involve NDA-covered or Restricted code?	Use local LLM only; close all other files	Internal LLM acceptable
Does the session involve credentials, API keys, or secrets?	Remove secrets from files before session	Proceed with caution
Is the repository classified as Confidential or Restricted?	Verify .rooignore is configured	Proceed

Step 2 — Verify tool configuration

- ☐ Confirm the AI tool is using the enterprise account (not a personal account)
- ☐ Confirm the tool is routing through the enterprise LLM proxy (LiteLLM endpoint), not directly to the cloud provider
- ☐ Verify .rooignore is present and up to date in the repository root
- ☐ Close all files not directly relevant to the current task
- ☐ Check that no .env files or credential files are open in the IDE

Step 3 — Set confirmation policy

- ☐ Confirm that "Always Allow" is **NOT** enabled for any tool category
- ☐ For agentic sessions (multi-step tasks): enable per-action confirmation for file writes, shell execution, and external network requests
- ☐ For simple completion sessions: standard confirmation policy is acceptable

Step 4 — After the session

- ☐ Review the agent's action log for unexpected file accesses or external requests

- ☐ Run secret scanning on any files modified by the agent (git diff | trufflehog or equivalent)
- ☐ If the agent accessed external URLs during the session, review those URLs for legitimacy

---

16.3 How-To: Repository Setup for AI-Safe Development

Configure every repository that will be used with an AI coding assistant using the following baseline controls.

Step 1 — Create .rooignore

Place a .rooignore file in the repository root. At minimum, include:

```
# Credentials and secrets
.env
.env.*
*.pem
*.key
*.p12
*.pfx
secrets/
credentials/

# Security-critical modules (customise per project)
src/crypto/
src/auth/
src/security/

# Configuration with connection strings
config/production/
config/staging/
appsettings.Production.json
appsettings.Staging.json

# Internal architecture documentation
docs/architecture/internal/
ADRs/internal/
```

Note: .rooignore syntax follows .gitignore conventions. Test the configuration by attempting to open an excluded file in the agent — it should be refused.

Step 2 — Configure pre-commit secret scanning

Install and configure a pre-commit hook to prevent secrets from being committed — including secrets that may have been introduced by AI-generated code:

```
# Install pre-commit and trufflehog
pip install pre-commit
pre-commit install

# .pre-commit-config.yaml
repos:
  - repo: https://github.com/trufflesecurity/trufflehog
    rev: v3.88.0
    hooks:
      - id: trufflehog
        name: TruffleHog
        entry: trufflehog git file://. --since-commit HEAD --only-verified --fail
        language: system
        pass_filenames: false
```

Step 3 — Add AI-generated code review gate

Add a CI/CD step that flags AI-generated code blocks for mandatory human review before merge:

```
# .github/workflows/ai-code-review.yml
- name: Flag AI-generated code
  run: |
    # Check for common AI generation markers in diff
    git diff origin/main...HEAD | grep -E "(Generated by|Co-authored-by: GitHub Copilot|AI-gene
rated)" \
      && echo "::warning::AI-generated code detected - mandatory security review required"
```

Step 4 — Repository classification label

Add a SECURITY.md or classification header to the repository README:

```
# AI Tool Policy

**Repository Classification:** [Public | Internal | Confidential | Restricted]

**Permitted AI Tools:**
- [ ] Cloud LLM (with ZDR agreement)
- [ ] Enterprise LLM Proxy (LiteLLM)
- [ ] Local LLM only
```

- [ ] No AI tools permitted

`.rooignore` status: Configured ✓ / Not configured ✗

16.4 How-To: MCP Server Installation and Vetting Checklist

Before installing any MCP server into a development environment, complete the following checklist. This applies to servers from public registries (npm, PyPI), GitHub repositories, and internal sources.

Step 1: Fork for Supply Chain Control

- ☐ **Fork the repository:** Before any other action, fork the third-party MCP server's official repository into the organisation's own version control system (e.g., internal GitLab).
- ☐ **Rationale:** This is the most critical step for supply chain security. It creates a controlled, internal copy that prevents silent updates or "rug pull" attacks from the original author. All subsequent vetting and deployment steps **must** be performed on this internal fork, not the public repository.

Step 2: Pre-installation Verification (on the Fork)

- ☐ **Publisher identity:** Verify the original publisher name matches the claimed organisation. Check the npm/PyPI profile for account age, other packages, and contact information. A newly created account publishing a high-profile integration is a red flag.
- ☐ **Source code review:** Read the forked server's source code (or at minimum the entry point and tool definitions). Look for: outbound HTTP calls to unexpected endpoints, base64-encoded strings, dynamic code evaluation (eval(), exec()), and environment variable exfiltration patterns.
- ☐ **Tool description review:** Read every tool description in the server manifest. Look for: unusual instructions, markdown formatting that could be interpreted as prompt injection, references to reading files or environment variables beyond the stated purpose.
- ☐ **Dependency audit:** Run npm audit or pip-audit on the server's dependencies. Reject servers with known critical vulnerabilities in dependencies.
- ☐ **Version history:** Check the original package's version history for sudden ownership changes, unusual version bumps, or removal of previous versions (a sign of a rug pull in progress).

Step 3: Installation Controls (from the Fork)

- ☐ **Pin the commit hash:** Do not deploy from a branch. Deploy from a specific, vetted commit hash from the internal fork.
- ☐ **Record the hash:** Record the deployed commit hash in a central registry.
- ☐ **Isolated test:** Test the server in an isolated environment (Docker container, VM) before deploying to a developer workstation with access to production credentials.
- ☐ **Minimal permissions:** Configure the server with the minimum OAuth scopes or API permissions required. For GitHub: use fine-grained PATs scoped to specific repositories, not classic PATs.

Step 4: Ongoing Monitoring

- ☐ Subscribe to security advisories for the original package (GitHub Dependabot, npm security advisories).
- ☐ **Do not auto-update:** When an update is available, create a pull request in the internal fork. The new code must go through the entire vetting process again (Steps 2-3).
- ☐ Implement startup hash verification: compare the running container/binary hash against the approved hash on every agent startup.
- ☐ Schedule quarterly re-validation of all installed MCP servers.
- ☐ Maintain an internal approved MCP server registry — only servers on the approved list may be used in production environments.

Red flags that require immediate rejection

Red Flag	Risk
Tool description contains <IMPORTANT>, [SYSTEM], or similar markup	Prompt injection attempt
Server makes outbound calls to domains not listed in documentation	Data exfiltration
Package published within 30 days of a high-profile MCP server launch	Impersonation attack
Source code contains process.env reads beyond stated credential needs	Credential harvesting
No source code available (binary-only distribution)	Cannot verify — reject

16.4.1 Solution Examples

Git, npm, and PyPI installation guides typically present the simplest configuration as the "Recommended" approach. The following example illustrates a pattern that is common in official documentation but represents an **anti-pattern** for secure corporate deployment — it installs directly from the public registry without version pinning, and embeds the credential inline:

```
// Anti-pattern: do NOT use in a corporate environment
{
 "mcpServers": {
  "paper-search-nodejs": {
   "command": "npx",
   "args": ["-y", "paper-search-mcp-nodejs"],
   "env": {
    "WOS_API_KEY": "your_web_of_science_api_key"
   }
  }
 }
}
```

The security-aware alternative installs from an **internal fork** of the repository, pins a specific **commit hash**, and externalises the credential via an environment variable reference — preventing both rug-pull attacks and accidental secret exposure:

```
{
  "mcpServers": {
    "paper-search-nodejs": {
      "command": "npx",
      "args": [
        "--node-options=--use-system-ca",
        "-y",
        "github:my-corp/paper-search-mcp-nodejs#<pinned-commit-hash>"
      ],
      "env": {
       "WOS_API_KEY": "${env:WOS_API_KEY}"
      }
    }
  }
}
```

Local versus centralised MCP deployment. The following two examples contrast a locally installed MCP server with a centrally deployed one. The first example connects to a centralised,

on-premise GitLab MCP server — eliminating the need to install the server on every developer machine. It also demonstrates the client-side disabledTools feature, which enforces a read-only posture at the agent level, and uses a credential with limited API access scope:

```
{
  "mcpServers": {
   "gitlab-node-full": {
      "_comment": "GitLab MCP — read access, write tools disabled client-side.",
      "type": "streamable-http",
      "url": "https://mcp-gitlab.my-corp.com/mcp?read",
      "headers": {
       "Authorization": "Bearer ${env:GITLAB_MCP_PAT_READ}"
      },
      "disabledTools": [
       "create_draft_note",
       "update_draft_note",
       "delete_draft_note",
       "..."
      ]
    }
  }
}
```

Limitation — blacklist vs. allowlist: The disabledTools mechanism is a **blacklist**: it blocks a named set of tools while permitting everything else by default. From a security standpoint, an **allowlist** (permit only explicitly named tools, deny all others) is the correct model — it ensures that newly added tools in a future server update are blocked until explicitly reviewed and approved. As of the time of writing, RooCode does not yet support a client-side allowedTools allowlist; disabledTools is the only available client-side tool-restriction mechanism. The LiteLLM MCP gateway (second example below) provides true allowlist enforcement at the proxy layer and is therefore the preferred control. [Project experience; see also RooCode issue #11259]

Project contribution: During this research, the InnovAIte project team actively engaged with the developers of coding agents — including RooCode — by reporting security limitations and requesting enhancements such as client-side allowlist support. This reflects the project's broader commitment to improving the security posture of the open-source AI coding assistant ecosystem, not only through internal controls but also through upstream contribution.

The second example shows an MCP server accessed through the LiteLLM MCP gateway, providing additional centralised governance, audit logging, and tool restriction at the proxy layer — addressing the blacklist limitation of client-side.

```
{
  "mcpServers": {
    "context7": {
      "_comment": "LiteLLM proxy MCP server — provides context7 library docs via the centra
lised gateway.",
      "type": "streamable-http",
      "url": "https://litellm.my-corp.com/context7/mcp",
      "headers": {
        "x-litellm-api-key": "Bearer ${env:LITE_LLM_API_KEY}"
      }
    }
  }
}
```

16.5 How-To: Configuring a Local LLM Proxy (LiteLLM) with Self-Signed Certificates

Enterprise environments frequently use internal Certificate Authorities (CAs) for HTTPS endpoints. When the LiteLLM proxy is deployed with a self-signed or internally-signed certificate, AI coding assistants (RooCode, Cursor, VS Code extensions) must be configured to trust that certificate. This section documents the configuration pattern discovered during the InnovAIte project. [Project experience]

Problem: RooCode and similar tools use Node.js/Electron HTTP stacks that do not automatically trust system certificate stores on all platforms. A self-signed certificate on the LiteLLM endpoint causes TLS handshake failures, leading developers to either disable TLS verification (dangerous) or fall back to personal cloud accounts (Shadow AI).

Solution: Trust the internal CA certificate

Step 1 — Export the internal CA certificate

Obtain the CA certificate in PEM format from your PKI administrator.

Step 2 — Configure Node.js to trust the CA (for Electron-based tools)

Set the NODE_EXTRA_CA_CERTS environment variable to point to the CA certificate file. This must be set *before* the IDE process starts:

```
# Windows (PowerShell — add to profile or system environment)
$env:NODE_EXTRA_CA_CERTS = "C:\path\to\certificate.pem"

# Linux/macOS (add to ~/.bashrc or ~/.zshrc)
export NODE_EXTRA_CA_CERTS="/path/to/certificate.pem"
```

Step 3 — Configure RooCode to use HTTPS endpoint

API Provider: LiteLLM API Base URL: https://litellm.internal.company.com:4000 API Key: < enterprise-api-key >

Critical: Always use https:// — never http://. An HTTP endpoint transmits all prompts (including sensitive source code) in plaintext over the network. If the self-signed certificate causes errors, the correct fix is to trust the CA certificate (Steps 1–2), not to downgrade to HTTP.

Step 4 — Verify the connection

```
# Test that the certificate is trusted
curl --cacert /path/to/certificate.pem https://litellm.internal.company.com:4000/health

# Expected: {"status": "healthy", ...}
# If you see certificate errors: the CA cert path is wrong or the cert has expired
```

Step 5 — LiteLLM proxy configuration for enterprise routing

A minimal LiteLLM config.yaml for routing between local models and governed cloud endpoints:

```
model_list:
  # Local model — for sensitive/confidential workloads
  - model_name: local-codellama
    litellm_params:
      model: ollama/codellama:34b
      api_base: http://localhost:11434

  # Governed cloud — for non-sensitive workloads (ZDR agreement required)
  - model_name: azure-gpt5
    litellm_params:
      model: azure/gpt-5
      api_base: https://company.openai.azure.com/
      api_key: os.environ/AZURE_API_KEY
      api_version: "2024-02-01"

general_settings:
  master_key: os.environ/LITELLM_MASTER_KEY
  # Audit logging — required for compliance
  store_model_in_db: true
  database_url: os.environ/DATABASE_URL
```

Security note: The master_key and all API keys must be stored as environment variables, never hardcoded in the config file. The config file itself should be excluded from version control (add litellm_config.yaml to .gitignore).

16.6 How-To: Incident Response When You Suspect AI-Mediated Data Leakage

If you suspect that sensitive source code or credentials have been exfiltrated via an AI coding assistant or MCP tool, follow this response procedure immediately.

Phase 1 — Contain (first 15 minutes)

1. **Revoke credentials immediately.** If API keys, tokens, or secrets may have been exposed: revoke them now, before investigating. Rotate: GitHub PATs, cloud provider API keys, database passwords, service account tokens. Do not wait for confirmation — revoke first, investigate second.
2. **Disconnect the agent.** Close the IDE or disable the AI tool. If using a local MCP server, stop the server process. If using a remote MCP server, revoke the OAuth token or API key used by the server.
3. **Preserve evidence.** Before closing anything: capture screenshots of the agent's action log, copy the conversation history, and export any available audit logs from the LiteLLM proxy or AI gateway.

Phase 2 — Investigate (first hour)

4. **Review the agent action log.** Look for: unexpected file reads (especially .env, credential files, files outside the current project), unexpected external HTTP requests, tool calls to repositories or APIs not related to the current task.
5. **Check network logs.** If a DLP proxy or AI gateway is deployed, review the logs for the session. Look for: large payload sizes (indicating bulk code transmission), requests to unexpected endpoints, DNS queries to unusual domains (DNS-based exfiltration pattern — see CVE-2025-55284, §5.2).
6. **Identify the scope of exposure.** Determine: which files were in the agent's context window, which external endpoints were contacted, which credentials were potentially exposed.

Phase 3 — Notify (within 4 hours)

7. **Notify the security team.** Provide: timestamp of the session, AI tool and version used, list of potentially exposed files, list of revoked credentials, network log evidence.

8. **Assess regulatory notification requirements.** If the exposed data includes personal data (GDPR), health data (HIPAA), or payment data (PCI-DSS): assess whether a breach notification is required. GDPR Article 33 requires notification to the supervisory authority within 72 hours of becoming aware of a breach.

Phase 4 — Remediate

9. **Audit the MCP server.** If the incident involved an MCP server: remove it immediately, review its source code for malicious behaviour, report to the package registry (npm, PyPI) if malicious.

10. **Update controls.** Add the exposed file types to .rooignore. Tighten OAuth scopes. Enable per-action confirmation for the tool category involved. Document the incident in the risk register.

Quick reference: Credential revocation URLs

Provider	Revocation URL
GitHub PAT	https://github.com/settings/tokens
GitHub Fine-Grained PAT	https://github.com/settings/personal-access-tokens
OpenAI API Key	https://platform.openai.com/api-keys
Anthropic API Key	https://console.anthropic.com/settings/keys
Azure OpenAI	Azure Portal → Cognitive Services → Keys and Endpoint
Google Cloud	https://console.cloud.google.com/apis/credentials

---

16.7 How-To: Deploying Sensitive-Data Detection and Centralised MCP Governance via LiteLLM

This section documents the two-layer governance architecture deployed during the InnovAIte project: (1) Qwen3Guard as a LiteLLM callback for inline sensitive-data detection, and (2) LiteLLM as a centralised MCP gateway. These two controls together address the gap between contractual ZDR agreements and technical enforcement. [Project experience]

16.7.1 Layer 1 — Qwen3Guard Sensitive-Data Detection Callback

Architecture overview:

Developer IDE (RooCode/Cursor)

|

▼ HTTPS (TLS with internal CA)

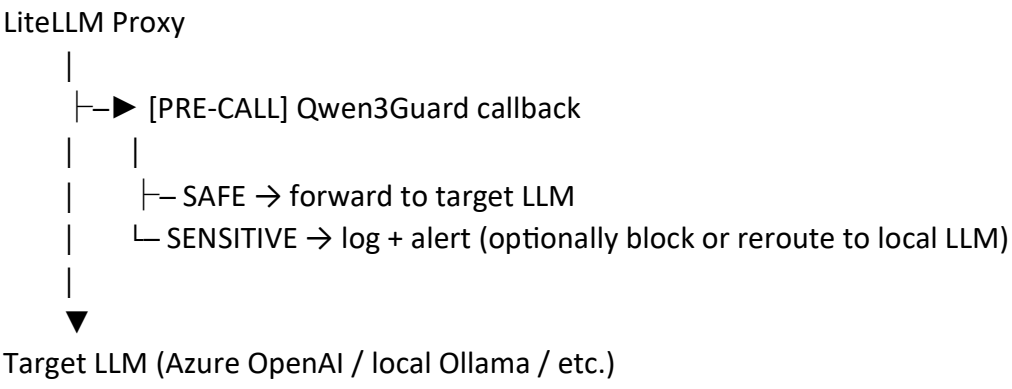


Figure 13: Qwen3Guard LiteLLM callback architecture (Section 16.7).

Step 1 — Deploy Qwen3Guard as a LiteLLM callback

LiteLLM supports custom callbacks via the callbacks configuration key. Implement a pre-call callback that invokes Qwen3Guard:

```
# litellm_callbacks/qwen3guard_callback.py
import litellm
from litellm.integrations.custom_logger import CustomLogger

class Qwen3GuardCallback(CustomLogger):
    """
    Pre-call callback: classifies prompt for sensitive data using Qwen3Guard.
    Logs flagged prompts; optionally blocks or reroutes them.
    """

    def __init__(self, guard_endpoint: str, threshold: float = 0.85):
        self.guard_endpoint = guard_endpoint  # Qwen3Guard inference endpoint
        self.threshold = threshold            # Sensitivity score threshold

    async def async_pre_call_hook(self, user_api_key_dict, cache, data, call_type):
        import httpx, json, logging

        messages = data.get("messages", [])
        prompt_text = " ".join(
            m.get("content", "") for m in messages if isinstance(m.get("content"), str)
        )

        async with httpx.AsyncClient() as client:
            response = await client.post(
                self.guard_endpoint,
```

```
            json={"text": prompt_text},
            timeout=5.0
        )
        result = response.json()

    score = result.get("sensitivity_score", 0.0)
    user_id = user_api_key_dict.get("user_id", "unknown")

    if score >= self.threshold:
        logging.warning(
            f"[Qwen3Guard] SENSITIVE prompt detected | user={user_id} | "
            f"score={score:.3f} | call_type={call_type}"
        )
        # Option A: Block the request
        # raise litellm.AuthenticationError("Sensitive data detected — request blocked")

        # Option B: Reroute to local model (add metadata for router)
        data["metadata"] = data.get("metadata", {})
        data["metadata"]["qwen3guard_flagged"] = True
        data["metadata"]["qwen3guard_score"] = score

    return data
```

Step 2 — Register the callback in LiteLLM config

```
# litellm_config.yaml
model_list:
  - model_name: azure-gpt5
    litellm_params:
      model: azure/gpt-5
      api_base: https://company.openai.azure.com/
      api_key: os.environ/AZURE_API_KEY

  - model_name: local-qwen3
    litellm_params:
      model: ollama/qwen3:32b
      api_base: http://localhost:11434

general_settings:
  master_key: os.environ/LITELLM_MASTER_KEY
  callbacks:
    - litellm_callbacks.qwen3guard_callback.Qwen3GuardCallback
```

```
  callback_settings:
    Qwen3GuardCallback:
      guard_endpoint: "http://qwen3guard.internal:8080/classify"
      threshold: 0.85
```

Step 3 — Configure routing: flagged prompts → local model

Add a router rule that sends flagged prompts to the local model:

```
router_settings:
  routing_strategy: usage-based-routing
  # Custom routing: if qwen3guard_flagged, use local model
  model_group_alias:
    sensitive-fallback: local-qwen3
```

In the callback (Step 1), when qwen3guard_flagged=True, set data["model"] = "sensitive-fallback" to trigger the reroute.

Step 4 — Verify detection is active

```
# Send a test prompt containing a synthetic credential
curl -X POST https://litellm.internal.company.com:4000/chat/completions \
  -H "Authorization: Bearer $LITELLM_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "azure-gpt5",
    "messages": [{"role": "user", "content": "Here is my API key: sk-test-FAKECREDENTIAL123"}]
  }'

# Check LiteLLM logs for Qwen3Guard warning:
# [Qwen3Guard] SENSITIVE prompt detected | user=... | score=0.923
# NOTE: score=0.923 is an illustrative example, not a measured detection result.
```

16.7.2 Layer 2 — LiteLLM as Centralised MCP Gateway

Architecture overview:

```
Developer IDE (RooCode/Cursor)
      |
      ▼ MCP over HTTPS (LiteLLM API key required)
LiteLLM MCP Gateway
      |
      ├─► Approved MCP Server A (e.g., GitHub MCP — scoped PAT)
```

├─► Approved MCP Server B (e.g., internal GitLab MCP)
└─► Approved MCP Server C (e.g., internal documentation search)

Network firewall: BLOCKS all direct MCP connections
(only LiteLLM endpoint is reachable on MCP ports)

Figure 14: LiteLLM centralised MCP gateway architecture for approved MCP servers (Section 16.7).

Step 1 — Register approved MCP servers in LiteLLM

```yaml
# litellm_config.yaml (MCP section)
mcp_servers:
  - server_name: github-mcp
    server_url: https://mcp.github.com
    auth_type: bearer
    auth_token: os.environ/GITHUB_MCP_TOKEN   # Fine-grained PAT, repo-scoped
    allowed_tools:
      - get_issue
      - list_pull_requests
      - get_file_contents
    # Explicitly blocked tools (defence-in-depth):
    blocked_tools:
      - create_repository
      - delete_repository
      - push_files

  - server_name: internal-gitlab-mcp
    server_url: https://gitlab01.hq.gratex.com/mcp
    auth_type: bearer
    auth_token: os.environ/GITLAB_MCP_TOKEN
    allowed_tools:
      - get_merge_request
      - list_issues
      - get_file

  - server_name: internal-docs-search
    server_url: https://docs.internal.company.com/mcp
    auth_type: none
    allowed_tools:
      - search_documentation
      - get_page
```

Step 2 — Configure developer API keys with MCP access

```
# Create a developer API key with access to approved MCP servers
litellm --config litellm_config.yaml &

curl -X POST https://litellm.internal.company.com:4000/key/generate \
 -H "Authorization: Bearer $LITELLM_MASTER_KEY" \
 -H "Content-Type: application/json" \
 -d '{
  "user_id": "developer@company.com",
  "allowed_mcp_servers": ["github-mcp", "internal-docs-search"],
  "metadata": {"team": "backend", "classification": "internal"}
 }'
```

Step 3 — Configure RooCode to use LiteLLM MCP endpoint

In RooCode MCP settings, replace direct MCP server URLs with the LiteLLM MCP gateway:

```
{
 "mcpServers": {
  "github": {
   "url": "https://litellm.internal.company.com:4000/mcp/github-mcp",
   "headers": {
    "Authorization": "Bearer <developer-litellm-api-key>"
   }
  },
  "docs": {
   "url": "https://litellm.internal.company.com:4000/mcp/internal-docs-search",
   "headers": {
    "Authorization": "Bearer <developer-litellm-api-key>"
   }
  }
 }
}
```

Step 4 — Firewall configuration

Block all direct MCP connections at the network perimeter. Allow only the LiteLLM endpoint:

```
# Firewall rule (conceptual — adapt to your network appliance)
ALLOW  developer-subnet → litellm.internal.company.com:4000  (HTTPS)
DENY   developer-subnet → *.github.com:443                   (direct MCP blocked)
DENY   developer-subnet → api.githubcopilot.com:443          (direct Copilot blocked)
DENY   developer-subnet → api.openai.com:443                 (direct OpenAI blocked)
```

```
DENY   developer-subnet → api.anthropic.com:443          (direct Anthropic blocked)
# Exception: allow browser traffic via proxy (separate rule)
```

Security note: The firewall rule blocking direct MCP connections is the critical enforcement layer. Without it, developers can bypass the LiteLLM gateway by configuring MCP servers directly in their IDE. The LiteLLM gateway provides governance; the firewall provides enforcement.

Step 5 — Verify the gateway is the only MCP path

```
# From a developer workstation, verify direct MCP is blocked:
curl -v https://mcp.github.com/sse 2>&1 | grep -E "Connection refused|timed out|blocked"
# Expected: connection refused or timeout (firewall blocking)

# Verify LiteLLM MCP gateway is reachable:
curl -H "Authorization: Bearer $DEV_API_KEY" \
  https://litellm.internal.company.com:4000/mcp/github-mcp/tools/list
# Expected: JSON list of allowed tools only (not the full GitHub MCP tool set)
```

Section provenance: Lessons LL-1 and LL-2 are grounded in the GitHub MCP Data Heist incident documented by Invariant Labs (May 2025) and analysed by Docker (2025). Lessons LL-3, LL-5, LL-7, and LL-8 reflect project-internal observations during InnovAIte AI tool integration. Lessons LL-4 and LL-6 combine project experience with OWASP guidance [§13.6, §7.3]. Lessons LL-9 and LL-10 document project-deployed controls: Qwen3Guard as a LiteLLM sensitive-data detection callback, and LiteLLM central MCP routing as a governance gateway — both implemented during the InnovAIte project [Project experience]. How-To sections 16.2–16.7 synthesise controls from §8.2 (Prevention), §9 (Security Strategies), §13 (Recommendations), and project-deployed architecture into actionable practitioner checklists.

---

17 Appendix A: Traceability to Description of Work

This appendix provides explicit traceability between the T6.3 task requirements, the D6.3 deliverable description, and the sections of this report that address each requirement.

17.1 A.1 T6.3 Task Requirements → Report Sections

T6.3 Requirement	Report Section(s)	Coverage
Prevention — hide/isolate sensitive modules while LLM manipulates the rest; restore afterwards	§8.2 Prevention and Isolation	Full — mechanism, semantic coherence challenge, evaluation methodology, open research questions, viability assessment
Obfuscation — transform sensitive code before LLM submission; restore afterwards	§8.3 Functional Obfuscation and Semantic Masking	Full — mechanism, quantitative de-obfuscation evidence, "Simplicity by Obfuscation" phenomenon, open research questions, viability assessment
Secure Channel — rely on dedicated secure channels provided by the LLM service	§8.4 Governed Secure Channel Architectures	Full — mechanism, critical limitations, verifiable ZDR research directions, open research questions, viability assessment
Thick Client — run the LLM locally so sensitive code never traverses external networks	§8.5 Thick Client — Localized Inference	Full — mechanism, hardware requirements, performance benchmarks, economics, technical security considerations, viability assessment
Better options? — whether there are better options than the four above	§8.6 Hybrid Architecture	Full — hybrid routing architecture as the researched "better" combined option
Which is most viable? — which of the above is most viable	§8.1 Comparative Framework, §13 Decision Framework	Full — comparative viability matrix + organisation-profile-based recommendations

17.2 A.2 D6.3 Deliverable Description → Report Sections

D6.3 Requirement	Report Section(s)	Coverage
"A document proposing approaches"	§8 (all paradigms), §8.6 (hybrid)	Full — all four paradigms plus hybrid architecture proposed
"Recommending approaches"	§13 Recommendations and Decision Framework	Full — organisation-profile-based decision framework with immediate actions

D6.3 Requirement	Report Section(s)	Coverage
"Dealing with security-critical resource submission to AI models"	§4 (architecture), §5 (threats), §6 (LCCT attacks), §7 (agentic threats)	Full — comprehensive threat and architecture analysis

17.3 A.3 Research Framing Clarification

The following table clarifies which findings in this report are **project conclusions** (synthesized from literature within this project) versus **external literature findings** (results from cited external studies):

Finding	Type	Source
AST-based SCM detection achieves ~88% F1-score	External literature	[39] Expert Systems with Applications, 2024
At 85% SCM accuracy, ~15% LLM utility loss	External literature	[40] ZeroSecBench, ICLR 2026
GPT-4o achieves 95.99% syntax accuracy and 93.42% execution accuracy (JsDeObsBench; 97.24% / 62.60% average across all models); GPT-4o and Claude 3.7 Sonnet assessed on a qualitative 0–5 scale (Tkachenko et al.)	External literature	[42, 43] JsDeObsBench, Tkachenko et al.
"Simplicity by Obfuscation" — LLMs simplify rather than obfuscate	External literature	[41] EASE 2025
ZDR agreements are contractual, not cryptographic	Project synthesis	Derived from [44] and provider documentation [this report §8.4.3]
Hybrid architecture is the emerging best practice	Project synthesis	Derived from [27] and threat analysis in §5–7
TEE.Fail compromises SGX/TDX/SEV-SNP at <$1,000	External literature	[36] TEE.Fail research, October 2025
Prevention/Isolation viability upgraded to HIGH	Project assessment	Based on RoguePilot [19] and threat analysis in §5–7
Computer-use/browser-use agents vulnerable to visual prompt injection (VPI-Bench: 51%–100% ASR)	External literature	[52] VPI-Bench, ICLR 2026 (peer-reviewed)

Finding	Type	Source
FigStep multimodal prompt injection achieves 82.5% ASR on LVLMs	External literature	[54] FigStep, AAAI-25 (peer-reviewed)
Agent guardrails are opaque: .cursorignore does not bind terminal/MCP tools	External literature	[68] Cursor documentation (vendor; non-peer-reviewed)
Fully-autonomous SE remains limited (SWE-bench 1.96% baseline)	External literature	[81] SWE-bench, ICLR 2024 (peer-reviewed)
SWE-bench contamination: 59.4% of audited test cases flawed	External literature	[31] industry analysis (non-peer-reviewed)
Open-weight models trail frontier by 3.3% on Chatbot Arena	External literature	[75] Stanford HAI AI Index 2026 (industry report)
Open-weight models reach ~85–95% of GPT-4 capability (aggregate estimate)	External literature	[9] IEEE BigData 2025 (peer-reviewed; aggregate estimate)
Local inference reaches ~98% of GPT-4 effective quality	External literature	[27] localaimaster.com (non-peer-reviewed commercial blog)
Coding-agent product landscape: OpenCode (MIT), Cline (Apache 2.0)	External literature	[71], [72] vendor documentation
Hidden auxiliary-agent model routing: title agent silently routes prompts to unexpected models	Project finding	[91] KiloCode source code analysis; [92] KiloCode PR #6733