BUILDING & OPERATIONALIZING SECURE AI

CYB-4203/6203: SECURE AND TRUSTWORTHY AI

Units 10.3, 10.4, 10.5 — Wednesday, April 22, 2026

Dallas Elleman — Spring 2026

Interactive version at
https://dallaselleman.github.io/cyb-4203-6203-spring-2026/course_materials/presentations/revealjs/pres-20.html

COURSE ORIENTATION

Section 3 — HOW

LAST SESSION — PRES 19

Red-Teaming AI Systems

*How to **break** AI/ML systems*

TODAY — PRES 20

Building & Operationalizing Secure AI

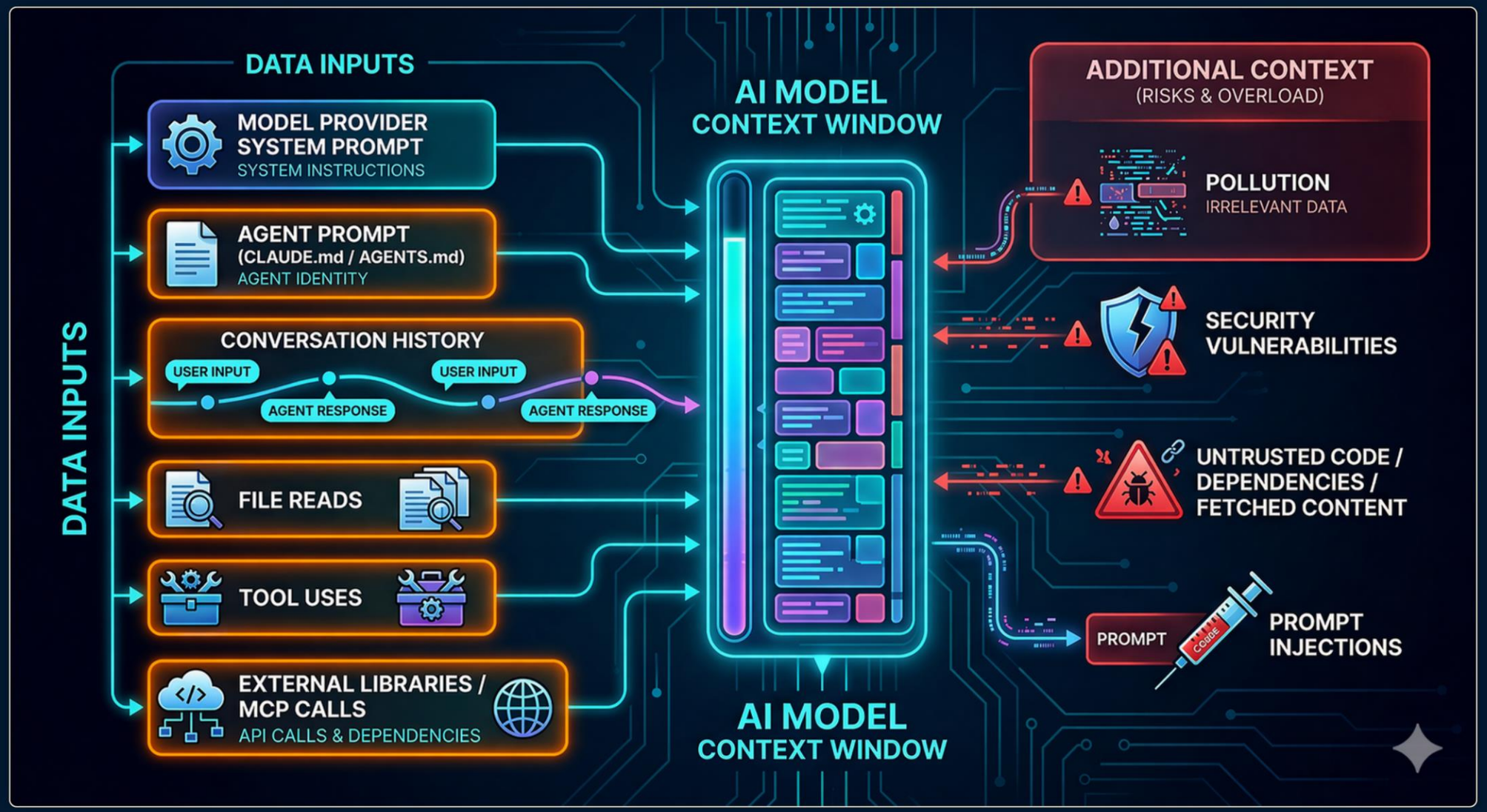
*How to **protect** AI/ML systems*

Focus: LLMs / Agentic Systems

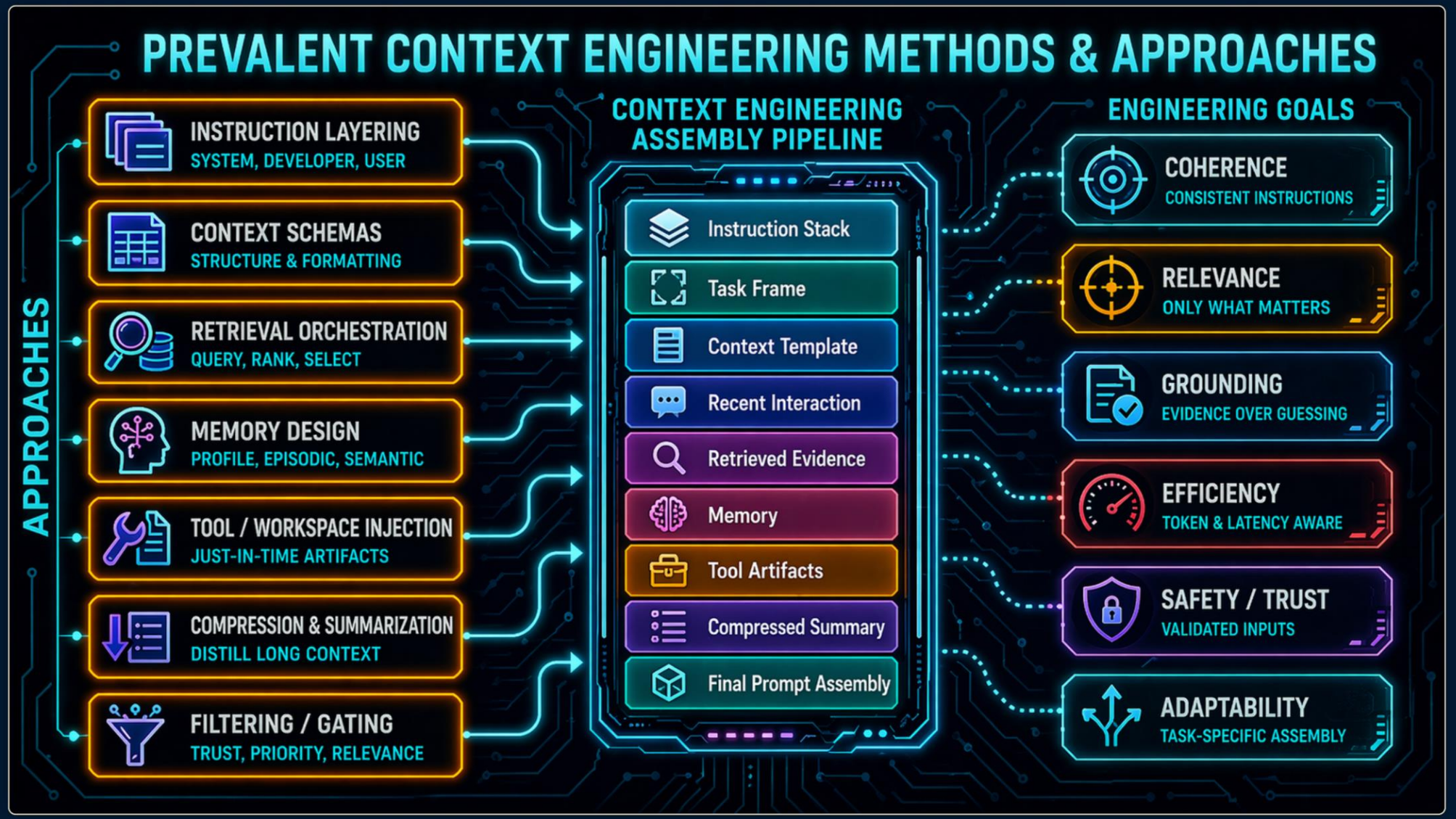
LLM CONTEXT ENGINEERING & SECURITY

Unit 10.3

CONTEXT WINDOW = TRUST BLENDER



CONTEXT ENGINEERING METHODS & APPROACHES



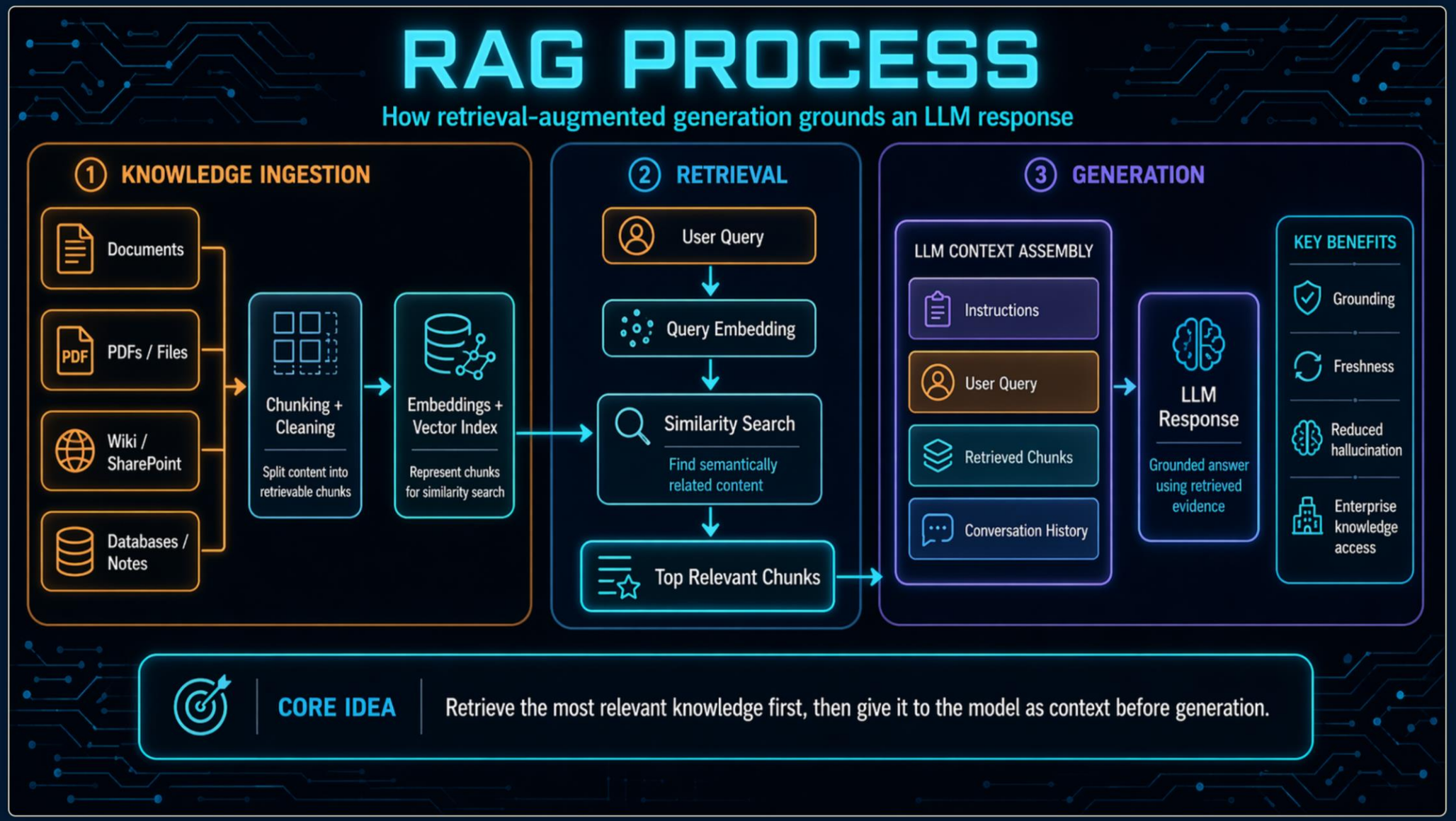
WHY CONTEXT ENGINEERING IS A SECURITY TOPIC

Context engineering methods such as RAG, vector stores, compaction and memory improve LLM/agentic **capabilities**.

From a security standpoint, each expands the set of inputs that LLMs treat as potentially authoritative.

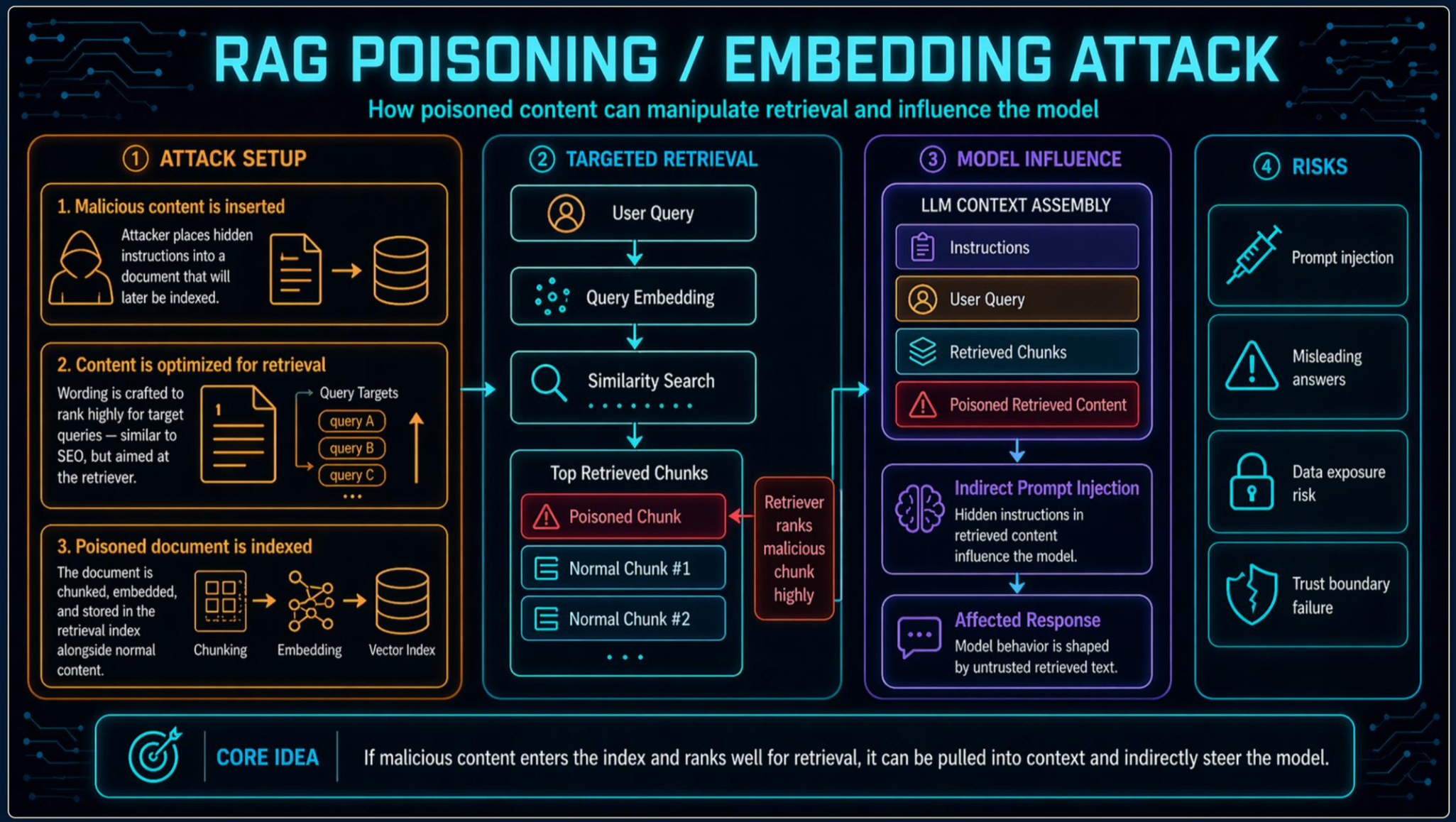
LLMs can't reliably distinguish between trusted and untrusted content on their own.

RETRIEVAL-AUGMENTED GENERATION (RAG)



User query → vector retrieval from a document store → retrieved chunks injected into the prompt → grounded answer.

RAG POISONING / EMBEDDING ATTACK



An attacker plants crafted content into the corpus — when retrieved, it becomes part of the model's prompt.

CASE STUDY: ECHOLEAK / CVE-2025-32711

First publicly-disclosed **zero-click prompt injection** against a production LLM system — Microsoft 365 Copilot. A single crafted email to a user caused Copilot to **exfiltrate internal data with zero user interaction**.

1. **XPIA classifier bypass** — prompt written as if aimed at a human, not an LLM, sneaking past the cross-prompt-injection-attack filter.

2. **Reference-style Markdown** to evade link redaction — the exfil URL was hidden in a reference definition rather than an inline link.

3. **Teams-proxy image URL** allowed by the content-security policy — the channel that actually exfiltrated the data out.

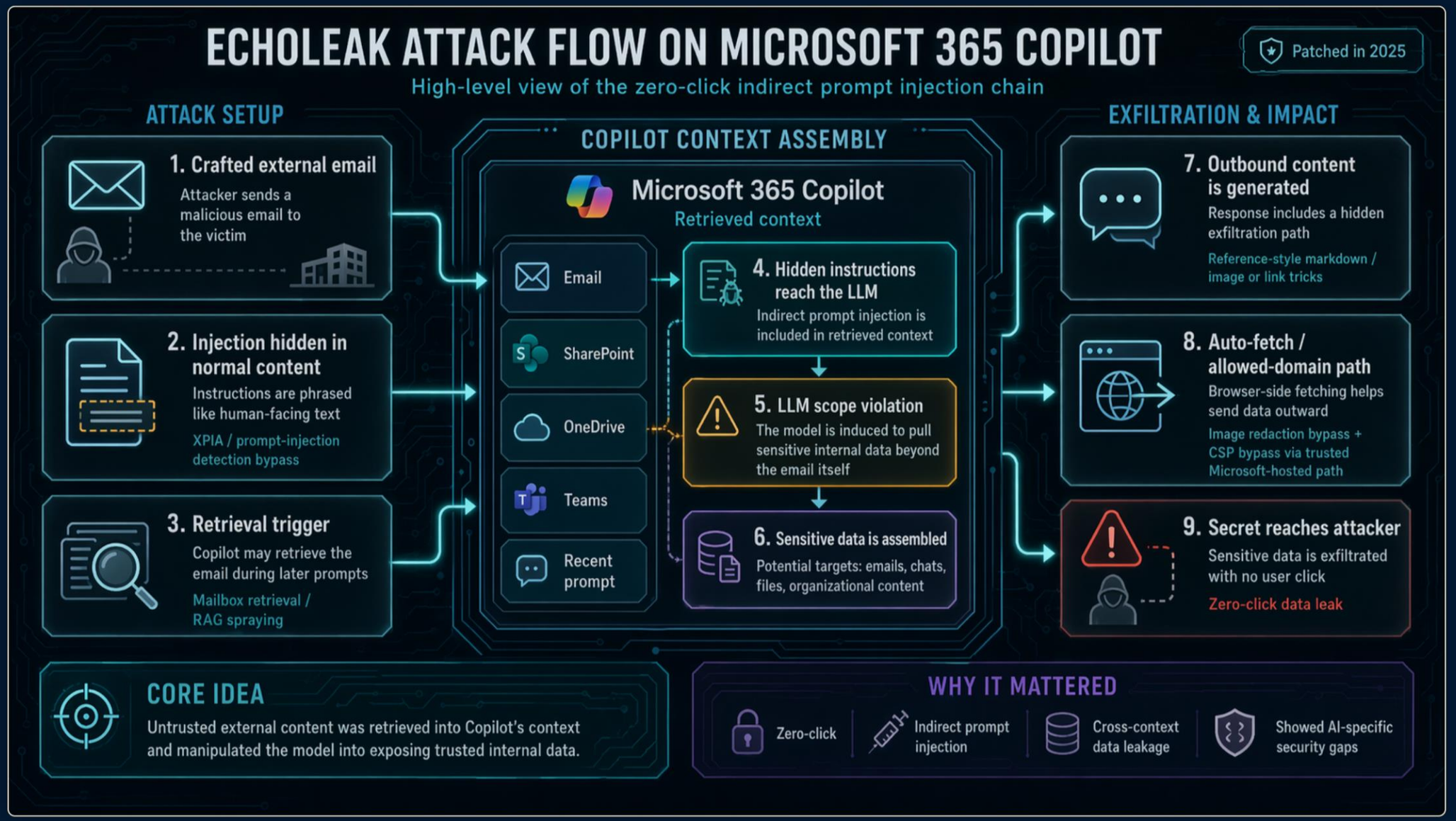
REFERENCES

Aim Labs / Aim Security disclosure — CVE write-up: thehackernews.com/2025/06/zero-click-ai-vulnerability-exposes.html

Academic case study — Reddy et al., arXiv:2509.10540: arxiv.org/abs/2509.10540

NIST NVD — nvd.nist.gov/vuln/detail/CVE-2025-32711

CASE STUDY: ECHOLEAK — EXPLOIT CHAIN



Crafted email → retrieval into Copilot's context → classifier bypass → Markdown-hidden link → Teams-proxy exfil.

GUARDRAILS AND DEFENSIVE MECHANISMS

Unit 10.4

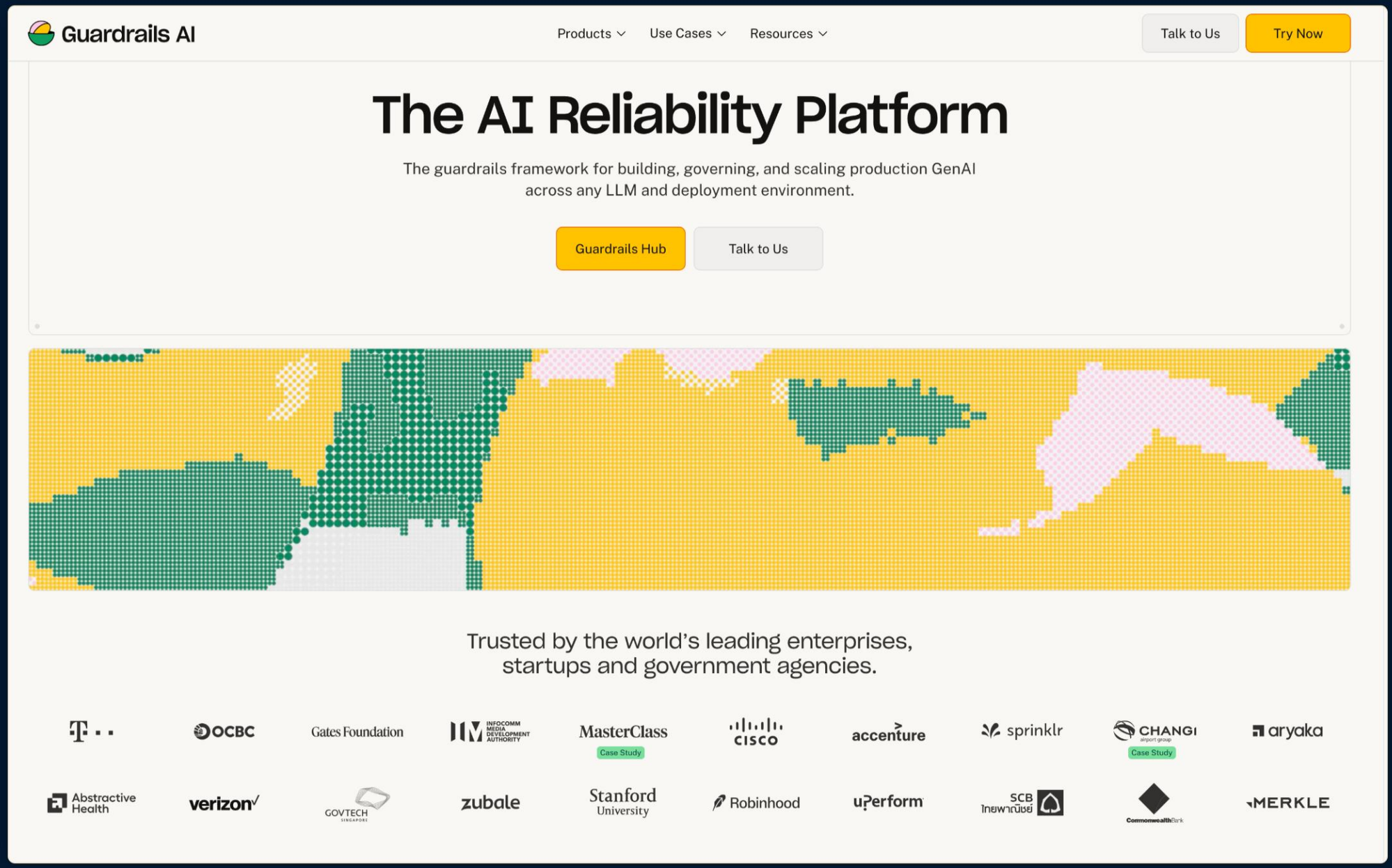
WHAT ARE GUARDRAILS?

Safety, security, and compliance constraints applied to Large Language Model inputs and outputs to prevent harmful, biased, or inappropriate behavior.

Use rule-based or AI-assisted checks to filter user prompts, data inputs, and model responses.

Absolutely essential for production systems in regulated industries.

TYPES OF GUARDRAILS



guardrailsai.com

MORE TYPES OF GUARDRAILS — NVIDIA NEMO

Use Case	Input	Retrieval	Dialog	Execution	Output
Content Safety	✅				✅
Jailbreak Protection	✅				
Topic Control	✅		✅		
PII Detection	✅	✅			✅
Knowledge Base / RAG		✅			✅
Agentic Security				✅	
Custom Rails	✅	✅	✅	✅	✅

developer.nvidia.com/nemo-guardrails

WHERE DO GUARDRAILS GO?

Input stage

Validate, classify, and sanitize prompts before they reach the model.

Retrieval / context stage

Enforce access on documents, filter tool responses, tag provenance.

Model stage

Safety-tuned models, conservative sampling, system-prompt hardening, token limits.

Tool-call / action stage

External authorization, per-action policy, approval gating.

Output stage

Content moderation, PII detection, structural validation, hallucination checks.

Execution stage

If the model produces code or actions, run them inside a sandbox.

Monitoring stage

Log everything, detect anomalies, trigger incident response.

REFERENCES

Iterathon — *AI Guardrails Production Implementation Guide 2026*: iterathon.tech/blog/ai-guardrails-production-implementation-guide-2026

AppSecEngineer — *How to Design Guardrails for Secure and Scalable AI Agents*: appsecengineer.com/blog/how-to-design-guardrails-for-secure-and-scalable-ai-agents

Wiz — *AI Guardrails*: wiz.io/academy/ai-security/ai-guardrails

RESTRICTED PERMISSIONS: META'S AGENTS RULE OF TWO

An AI agent should satisfy at most two of the following three properties in a single session:

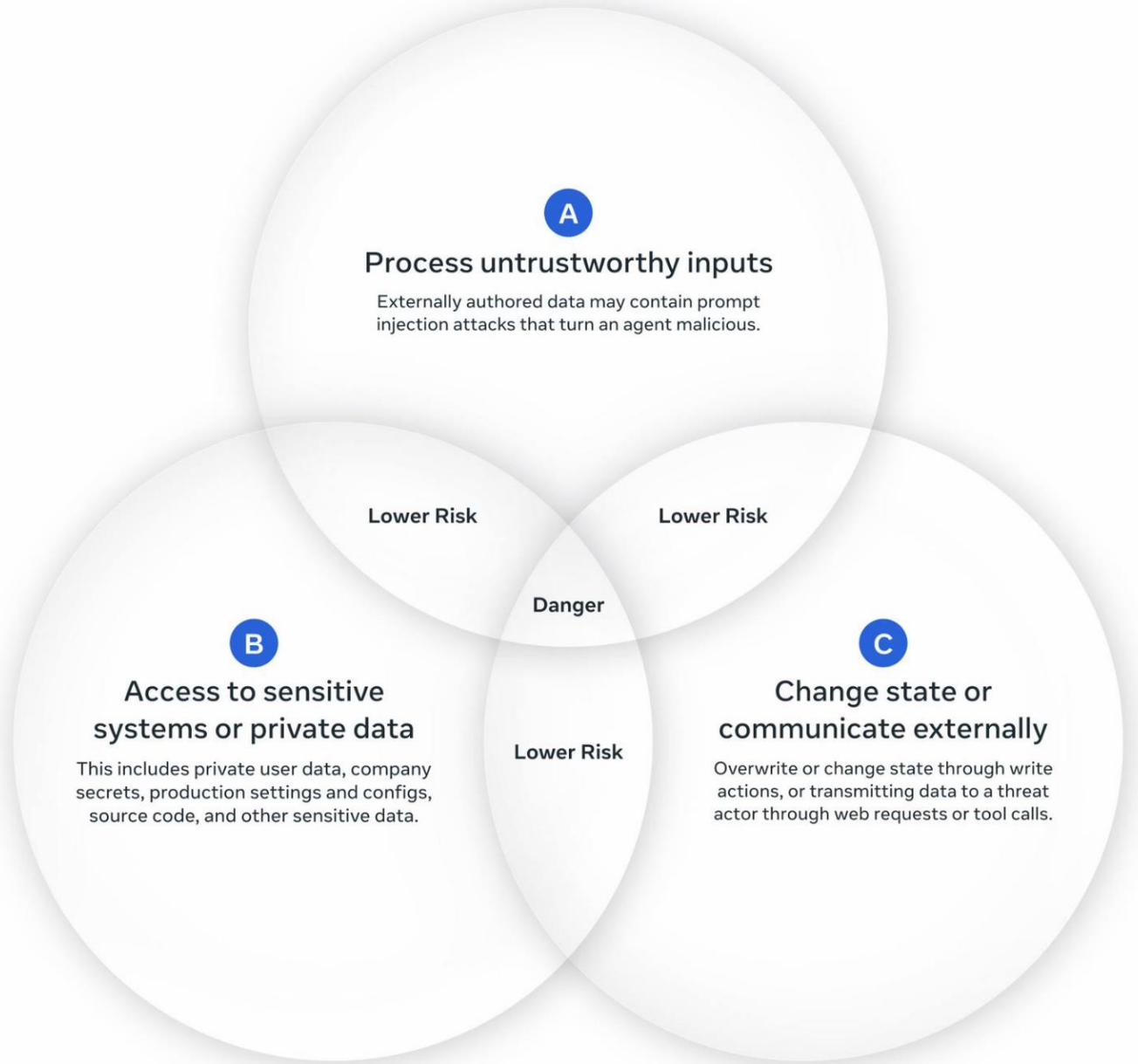
A Processes **untrusted inputs**.

B Has access to **sensitive systems or private data**.

C Can **change state or communicate externally**.

If all three are required for a task, the agent must restart the session or require human-in-the-loop approval before acting.

Choose Two



simonwillison.net/2025/Nov/2/new-prompt-injection-papers

META'S AGENTS RULE OF TWO — EXAMPLES

Travel assistant

A (web) + **B** (user data) allowed, but **C** (booking/payment) is gated behind human confirmation and restricted to URLs from trusted sources.

Browser research agent

A (web) + **C** (arbitrary requests) allowed, but **B** is eliminated by running in a session-free sandbox with no preloaded credentials.

Engineering agent

B (production access) + **C** (stateful changes) allowed, but **A** is controlled by filtering to trusted data sources via author-lineage.

REFERENCES

Meta AI — *Agents Rule of Two: A Practical Approach to AI Agent Security* (31 Oct 2025): ai.meta.com/blog/practical-ai-agent-security

Simon Willison's review: simonwillison.net/2025/Nov/2/new-prompt-injection-papers

OUTPUT FILTERING (1 OF 2)

Assume the model will produce something unsafe eventually, and design for it.

Structural validation

Pydantic schemas, JSON schema validation. Catches malformed output before it crashes downstream systems. (Usually the *most common* real failure mode.)

Content moderation

Toxicity, hate speech, self-harm, NSFW — via purpose-built classifiers (e.g., Azure Content Safety, Lakera Guard).

PII detection and redaction

Enforced *after* generation, not only before input. Models can regurgitate PII from context even if the user didn't ask for it.

Harmful-content blocking

Refuse or rewrite outputs that provide dangerous instructions (CBRN, malware, fraud).

OUTPUT FILTERING (2 OF 2)

Groundedness / hallucination checks

For RAG, verify claims are supported by the retrieved documents.

Data-exfiltration guards

Redact URLs, strip Markdown images, flag outbound links not from an allowlist. (The single most effective mitigation against trifecta exploits.)

Format contracts

No rendering of arbitrary HTML or JavaScript. Link sanitization. No auto-loaded images pulling from attacker-controlled URLs.

REFERENCES

ToolHalla — *AI Agent Guardrails & Output Validation in 2026*: toolhalla.ai/blog/ai-agent-guardrails-io-validation-2026

Microsoft Foundry — *Guardrails and controls overview*: learn.microsoft.com/en-us/azure/foundry/guardrails/guardrails-overview

Cloud provider controls: Guardrails for Amazon Bedrock, Azure OpenAI content filters, Google Vertex AI safety filters.

SANDBOXING (1 OF 2) — WHY

Agents that generate and execute code must run that code in an environment where compromise is survivable.

Traditional sandboxes (Docker, bubblewrap, macOS Seatbelt, Windows AppContainer) share the host kernel. Any kernel vulnerability becomes an escape hatch. For code the model generates on-the-fly — which, by construction, you haven't reviewed — that is not good enough.

NVIDIA's AI Red Team guidance (Jan 2026) is explicit: **fully virtualized environments (VMs, unikernels, Kata) should be the default for agentic code execution, not shared-kernel containers.**

SANDBOXING (2 OF 2) — SPECTRUM OF ISOLATION

WEAKEST → STRONGEST

Shared-kernel containers

Docker defaults — lightweight, fast; *not* suitable for arbitrary untrusted code.

gVisor

User-space kernel that mediates syscalls. Better than shared-kernel; still a shared attack surface.

MicroVMs

Firecracker, Kata Containers, Cloud Hypervisor — dedicated kernel per workload, hardware-enforced isolation, ~100–200 ms cold start.

V8 isolates / Dynamic Workers

Millisecond startup; smaller than VMs but a more complicated hardening story (V8 has more security bugs than typical hypervisors).

SANDBOXING: NETWORK & SECRET HYGIENE (1 OF 2)

Filesystem isolation

The agent should not be able to read SSH keys, `.env` files, shell rc files, or paths in $PATH.

Network isolation

Without it, a compromised agent can exfiltrate anything it can read. Default deny outbound; allow-list specific hosts if absolutely needed.

No secrets inside the sandbox

API keys, tokens, DB credentials should never be injected as env vars or mounted files. The sandbox protects you from the agent, but it does not protect the sandbox's contents from a prompt-injected agent. Keep secrets outside and expose only narrow, parameterized tool interfaces.

SANDBOXING: NETWORK & SECRET HYGIENE (2 OF 2)

Lifecycle management

Fresh sandbox per task when possible. Ephemeral filesystems. Clear TTLs. Don't let credentials, code, or IP accumulate across sessions.

Capability-based access, not ambient permissions

Expose specific, narrowly-defined operations; do not expose a shell and hope for the best.

REFERENCES

NVIDIA — *Practical Security Guidance for Sandboxing Agentic Workflows*: developer.nvidia.com/blog/practical-security-guidance-for-sandboxing-agentic-workflows

Northflank — *How to sandbox AI agents in 2026*: northflank.com/blog/how-to-sandbox-ai-agents

Kubernetes Agent Sandbox SIG: agent-sandbox.sigs.k8s.io

Claude Code sandboxing: code.claude.com/docs/en/sandboxing

Cloudflare — *Sandboxing AI agents, 100x faster*: blog.cloudflare.com/dynamic-workers

LangChain — *Sandboxes* docs: docs.langchain.com/oss/python/deepagents/sandboxes

HUMAN OVERSIGHT

Human-in-the-loop (HITL)

A human must actively approve before the agent acts on specific decisions. Synchronous checkpoint. Used for high-stakes, irreversible, or ambiguous actions.

Human-on-the-loop (HOTL)

A human supervises overall behavior through dashboards and alerts, intervening on anomalies but not on individual actions. Asynchronous supervision.

Human-out-of-the-loop

Fully autonomous operation. Appropriate for low-risk, high-volume, reversible actions only. Still needs logs and sampling audits.

REFERENCE

Strata — *Practicing the Human-in-the-Loop*: strata.io/blog/agentic-identity/practicing-the-human-in-the-loop

HUMAN OVERSIGHT SCREENING (1 OF 2)

Simple four-question screen: if the answer to any is yes, that workflow belongs behind an in-loop gate before the agent acts.

1. Is the decision **irreversible**? (financial transactions, data deletion, production deploys)
2. Does the agent have **write access** to production systems or financial flows?
3. Are the consequences **material** for customers or regulators? (EU AI Act high-risk, HIPAA, FINRA, GDPR)
4. Is the task **ethically sensitive or novel**? (protected categories, out-of-distribution)

HUMAN OVERSIGHT SCREENING (2 OF 2)

DESIGN ANTI-PATTERNS TO WATCH FOR

Automation complacency

Reviewers rubber-stamp approvals because the system is usually right. Mitigate with audits, rotation, spot-checks.

Review bottleneck

Too many approvals per unit time — queue grows, system stalls or reviewers rubber-stamp. Keep escalation rates ~10–15%.

Reviewer without context

"Approve tool call: send_email?" is useless. Show full intent, arguments, diffs, predicted outcomes.

REFERENCES

Elementum AI — *Human-in-the-Loop Agentic AI*: elementum.ai/blog/human-in-the-loop-agentic-ai

Galileo — *How to Build Human-in-the-Loop Oversight for AI Agents*: galileo.ai/blog/human-in-the-loop-agent-oversight

LangChain human-in-the-loop middleware: docs.langchain.com/oss/python/langchain/human-in-the-loop

FINAL PROJECT

COMING TONIGHT

I'll complete the **Final Project assignment** and **release it this evening**.

Watch your email / the course site — full spec, teams, rules of engagement, and OpenClaw environment access will go out together.