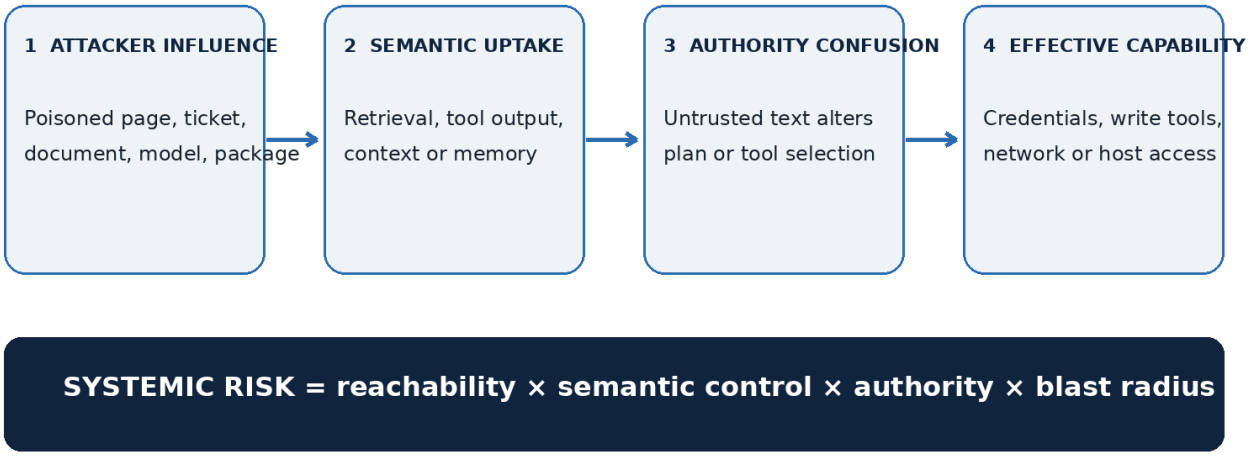
ARCHITECTURE-LEVEL ATTACK VECTORS AGAINST LLM SYSTEMS

An evidence-led threat model for RAG, vector stores, agents, tools, and the AI supply chain

How architecture-level attacks compose



Advanced technical edition • Research cut-off: 26 July 2026

Contents

An evidence-led threat model for RAG, vector stores, agents, tools, and the AI supply chain

Audience: AI security engineers, platform architects, red teams, security leadership, and technical risk owners Research cut-off: 26 July 2026 Scope: Production systems built around large language models (LLMs), retrieval-augmented generation (RAG), text embeddings, autonomous tool use, model repositories, Python dependencies, and Model Context Protocol (MCP) integrations.

> Safety boundary. This report is defensive. Attack descriptions are deliberately architectural rather than operational. It does not provide payloads, credential-theft procedures, or exploit code.

Executive summary

The central security problem is not that an LLM has become a conventional code-execution engine. It is that a probabilistic model is increasingly placed inside a deterministic control plane: it reads untrusted content, selects tools, carries credentials, and influences irreversible state. Indirect prompt injection, embedding inversion, excessive agency, and supply-chain compromise are therefore best understood as composable failures of trust boundaries rather than four isolated bugs.

Indirect prompt injection is the best-demonstrated architecture-level vector. The foundational real-world study showed that instructions planted in data later consumed by an LLM-integrated application could redirect behavior, influence API calls, and enable data theft or persistent contamination (Greshake et al., 2023). Subsequent benchmarks confirm that the risk survives beyond demonstrations: InjecAgent contains 1,054 cases across 17 user tools and 62 attacker tools, and reported a 24% attack success rate against a ReAct-prompted GPT-4 agent in its tested configuration (Zhan et al., 2024). AgentDojo adds 97 realistic tasks and 629 security test cases, while also showing an uncomfortable operational truth: useful task completion and security robustness are both difficult to preserve simultaneously (Debenedetti et al., 2024).

Embedding inversion is also demonstrated, but the phrase “reconstruct text from vectors alone” needs qualification. Vec2Text recovered 92% of 32-token inputs exactly in a particular experimental setup, using a learned iterative decoder and query access to the target embedding model; it is not evidence that any arbitrary vector database can be decoded without knowing or querying the encoder (Morris et al., 2023). A separate generative inversion study recovered coherent, semantically similar sentences and exceeded earlier inversion methods, reinforcing that embeddings should not be treated as anonymized text (Li, Xu & Song, 2023). The security implication is still serious: compromise of embeddings, encoder access, and relevant auxiliary data may become a confidentiality incident even when raw documents are stored elsewhere.

Agentic tool abuse is the impact multiplier. OWASP defines excessive agency as damaging action enabled by excessive functionality, permissions, or autonomy, including when a model is manipulated by indirect injection (OWASP, 2025). MCP expands the integration surface and formalizes tool discovery and invocation, but its own security guidance warns about confused-deputy behavior, token passthrough, server-side request forgery, session hijacking, and local-server compromise (Model Context Protocol, 2026). The architecture must therefore treat model output as an untrusted proposal, not authorization.

Supply-chain compromise is partly novel and partly familiar. A poisoned model checkpoint can exploit unsafe deserialization; Hugging Face explicitly warns that pickle-based model loading can execute arbitrary code and that its scanning is not foolproof (Hugging Face, 2026). PyTorch 2.6 changed `torch.load` to use `weights_only=True` by default when no custom pickle module is supplied, narrowing—without eliminating—the risk (PyTorch documentation). Malicious dependencies and local MCP servers remain conventional code-execution supply-chain risks, but they become more dangerous in AI workstations because the compromised component may inherit model-provider keys, repository tokens, browser sessions, local files, and tool credentials.

The defensible pattern is a small trusted control plane around an untrusted semantic engine: provenance-tag every input; isolate retrieval content from policy; minimize tool scopes; enforce authorization outside the model; require confirmation for high-impact actions; constrain network egress; sandbox models and local servers; pin and attest artifacts; and record causal traces that link evidence, decisions, tool calls, identities, and effects. No prompt template creates a hard security boundary. Architectural controls must remain effective even when the model follows the attacker’s text.

Key findings

RAG is a delivery channel, not a security boundary. Retrieval improves relevance, but it also selects attacker-controlled text for placement in a privileged context. A similarity score says nothing about the authority or safety of the retrieved content (Greshake et al., 2023).

Embeddings are sensitive derivatives, not anonymous data. Published inversion methods can reconstruct full or semantically equivalent text under meaningful assumptions; encryption, isolation, retention controls, and breach analysis should cover vectors as well as source documents (Morris et al., 2023; Li, Xu & Song, 2023).

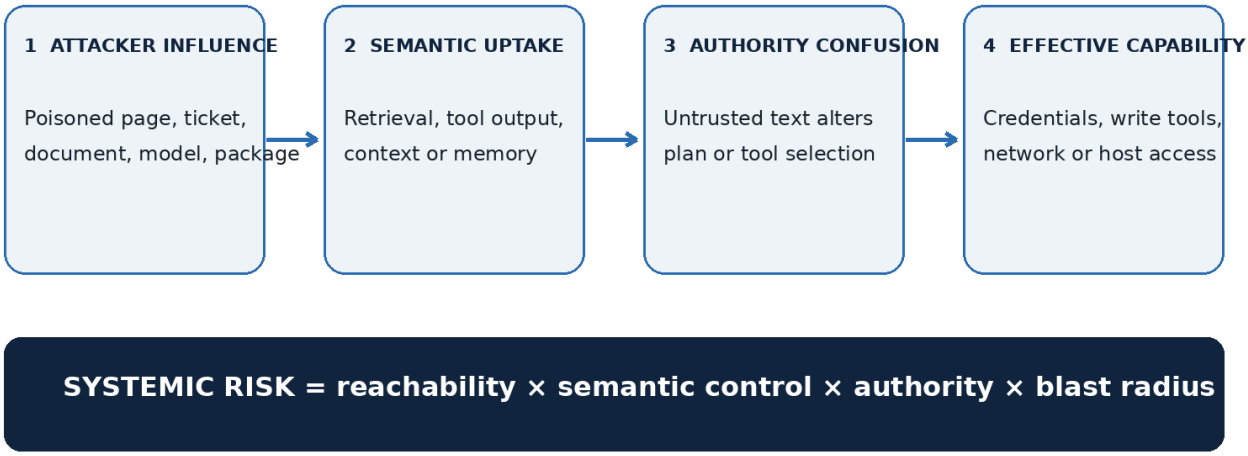
Agency determines blast radius. An injection into a read-only summarizer may corrupt an answer; the same injection in an agent with mail, repository, payment, or infrastructure tools can become an integrity or confidentiality event (OWASP, 2025).

Prompt-only defenses are insufficient. Benchmarks show that attacks and defenses co-evolve, and defenses can reduce utility or fail against adaptive attacks (Debenedetti et al., 2024; Yi et al., 2025).

Supply-chain controls must cover more than weights. Model files, tokenizer code, configuration, remote-code flags, inference containers, packages, agent skills, MCP server commands, and hosted endpoints all cross different trust boundaries (Hugging Face security; MCP security guidance).

The four vectors compose. A poisoned dependency or MCP server can inject content; injected content can make an overprivileged agent query secrets; vector leakage can reveal material used to craft higher-fidelity injections. Risk reviews that score each component independently will understate system risk. This compositional assessment is Ask Anything analysis based on the cited attack mechanisms.

How architecture-level attacks compose



Original illustration: Ask Anything threat synthesis. Evidence basis: Greshake et al., OWASP Excessive Agency, and MCP Security Best Practices.

1. Threat model: the semantic control plane

1.1 The assets and principals

A production LLM system normally contains more than a model endpoint. It includes system and developer instructions, conversation history, retrieved documents, embedding indexes, model and tokenizer artifacts, tool schemas, short- and long-term memory, credentials, external APIs, and human approval flows. Attackers may control a document author, website, support ticket, email sender, repository contributor, package publisher, model-repository account, MCP server, or compromised upstream maintainer.

The relevant security principals are therefore: the end user; the application operator; content authors; data owners represented in the index; tool and API owners; model or package publishers; and the model itself.

The model is not a principal that can safely confer authority. It is a decision-support component whose output may be attacker-influenced.

1.2 Why “architecture-level” is useful—but incomplete

These vectors share a system-design property: the dangerous outcome appears at a boundary around the model, not necessarily inside its weights. Prompt injection exploits mixed-trust context; inversion exploits a representation interface; tool abuse exploits the action interface; supply-chain poisoning exploits artifact and execution trust. Yet supply-chain compromise can still be ordinary arbitrary code execution through deserialization or package installation. Calling every case “architecture-level” should not obscure mature controls such as sandboxing, signing, reproducible builds, least privilege, and egress filtering.

MITRE ATLAS separately enumerates LLM prompt injection, RAG poisoning, AI-agent tool poisoning, AI supply-chain compromise, credential harvesting, and tool invocation, supporting a system-wide view rather than a model-only view (MITRE ATLAS). This taxonomy is useful for coverage, but its presence does not establish prevalence or incident frequency.

1.3 A composable kill chain

A high-impact chain typically needs four conditions:

Attacker influence: malicious text or artifact enters a reachable source.

Semantic uptake: retrieval or a tool places it where the model processes it.

Authority confusion: the model treats content as an instruction or selects an attacker-favored action.

Effective capability: downstream software executes the action with useful credentials, network access, or write permission.

Breaking any one condition can stop a specific chain. Reliable systems break several: content provenance and retrieval hygiene at entry; instruction/data separation and attack evaluation at inference; a deterministic policy decision point before tools; and sandbox, egress, and identity controls at execution. This is Ask Anything analysis, not a claim that a universal four-stage standard exists.

2. Indirect prompt injection through RAG and external content

2.1 Mechanism

In direct prompt injection, the user supplies the adversarial instruction. In indirect injection, an attacker plants it in content the application may later retrieve or inspect. The model receives trusted instructions and untrusted data in one token stream, and learned instruction-following behavior can cause the untrusted segment to alter the task. Greshake and colleagues demonstrated this across web content and LLM-integrated applications, including impacts involving API control, data theft, and self-propagating contamination (Greshake et al., 2023).

RAG adds two useful attacker levers. First, the attacker can optimize text to be retrieved for predictable queries. Second, retrieved passages often arrive close to the user request and may be framed as “context,” making them salient to the model. The vector database is not executing the prompt; it is ranking and transporting the malicious text. The failure occurs when the application treats retrieval relevance as authorization.

Injection may be visible prose, white-on-white text, metadata, comments, OCR text, code comments, or a tool response. Hidden rendering is not required. Defensive scanners that search for phrases such as “ignore previous instructions” address only a brittle subset because the instruction can be paraphrased, split across sources, encoded, or presented as a workflow rule.

2.2 Evidence strength and limitations

Evidence is strong that indirect injection is feasible and moderate on how well specific mitigations generalize. BIPIA was designed to benchmark indirect injection and reported broad vulnerability across tested models; its proposed boundary-awareness and explicit-reminder defenses substantially reduced attacks, with the white-box approach reaching near-zero attack success in that benchmark (Yi et al., revised 2025). That result is a benchmark result, not a proof of security against adaptive attackers.

InjecAgent's 1,054 cases span 17 user tools and 62 attacker tools; its reported 24% success rate for ReAct-prompted GPT-4 provides a concrete measure under one experimental configuration (Zhan et al., 2024). AgentDojo's 97 tasks and 629 security cases explicitly measure both benign utility and security, and found that current attacks do not break every security property while current models also fail many benign tasks (Debenedetti et al., 2024). The conflicting-looking outcomes are compatible: success rates depend heavily on model, agent loop, tool surface, attack objective, scoring rule, and whether the attacker adapts to the defense.

2.3 Documented examples

• Web and code-assistant demonstrations: the foundational study tested indirect injections against real LLM-integrated applications and synthetic tool-enabled systems, documenting remote control through content without direct access to the user prompt (Greshake et al., 2023).

• Cross-tool harm and exfiltration: InjecAgent operationalized attacks such as direct user harm and private-data exfiltration across mail, cloud, and other tool classes (Zhan et al., 2024).

• Banking, travel, email, and workspace tasks: AgentDojo provides reproducible environments in which untrusted tool data can attempt to redirect agents in realistic workflows (AgentDojo project).

• Session-event injection in MCP: official MCP guidance describes a distributed-server flow in which a malicious event associated with a stolen session identifier can be delivered to the legitimate client and influence behavior (MCP Security Best Practices).

These are documented studies and protocol attack descriptions, not a claim that each produced a publicly confirmed enterprise breach.

2.4 Controls

At ingestion: preserve author, origin, timestamp, trust level, transformations, and access-control metadata. Scan and quarantine suspicious content, but do not make scanning the sole boundary. Prevent low-trust tenants from writing into high-trust indexes. Apply access control before retrieval and again before generation.

At retrieval: use separate indexes or namespaces for different trust zones; restrict the number and size of retrieved chunks; carry provenance into the prompt; and avoid retrieving instructions, configuration, or secrets as ordinary text. Retrieval filters reduce exposure but cannot prove that benign-looking text is safe.

At inference: delimit untrusted content and instruct the model not to follow it. These measures can improve robustness, as BIPIA suggests, but must be treated as probabilistic defenses (Yi et al., 2025). Use a second-stage classifier or policy model as telemetry, not as sole authorization.

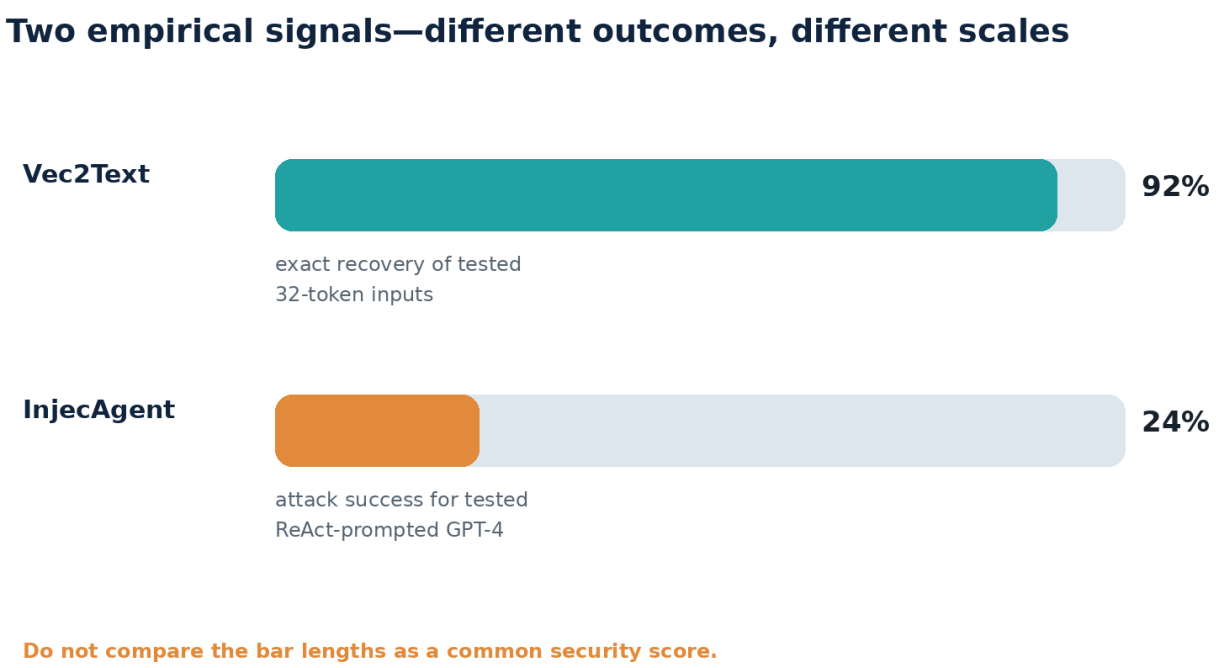
At action: bind every tool call to a user intent, identity, object, and allowed effect. Require deterministic validation and step-up approval when the call sends data, changes access, spends money, executes code, or modifies production. Suppress secrets from model context and tool returns unless indispensable.

At network: default-deny egress for agent runtimes and allow only task-required destinations. MCP guidance recommends URL and redirect validation, IP-range restrictions, and egress proxies to reduce server-side request forgery exposure (MCP Security Best Practices).

3. Vector embedding inversion

3.1 What has been demonstrated

Text embeddings are lossy representations optimized for semantic relationships, but "lossy" does not mean "private." Vec2Text frames inversion as iterative controlled generation: produce a text hypothesis, embed it, compare it with the target vector, and correct the hypothesis. The paper reports exact recovery of 92% of 32-token inputs in its tested setup and recovery of full names from clinical-note data (Morris et al., 2023). Li, Xu, and Song independently trained a generative decoder and reported coherent, contextually similar reconstructions that outperformed prior methods on their evaluation measures (Li et al., 2023).



Source: reconstructed chart from Morris et al., 2023 (92% exact recovery of 32-token inputs in the reported Vec2Text setup) and Zhan et al., 2024 (24% attack success for ReAct-prompted GPT-4 in the reported InjecAgent setup). The studies measure different phenomena; bars must not be compared as a common performance scale.

3.2 Correct threat assumptions

The strongest result did not show universal recovery from an unexplained vector in isolation. The adversary used a decoder trained for target embedding models and query access to re-embed hypotheses (Morris et al., 2023). Reconstruction quality can vary with encoder, text length, domain, pooling, quantization, attacker training data, query access, and whether the stored vector represents one sentence, a chunk, or an aggregate.

Accordingly:

• Demonstrated: short text can be exactly reconstructed at high rates under a favorable, model-specific setup; sensitive attributes can leak; semantically similar sentences can be generated.

• Plausible but environment-dependent: proprietary passages, code fragments, or credentials might be recovered if they are embedded as short, distinctive strings and the attacker has the necessary encoder access and auxiliary capability.

• Not established by the cited studies: reliable near-exact recovery of arbitrary long documents, arbitrary proprietary embedding models, or every credential from a vector database “alone.”

This distinction matters operationally. A stolen vector index plus public access to the same embedding API may satisfy more assumptions than a stolen index built with a private, isolated encoder. Conversely, even imperfect semantic reconstruction can disclose topic, identity, diagnosis, project name, or business intent.

3.3 Controls

Treat embeddings as data in the same classification tier as their source unless a documented privacy assessment supports a lower tier. Encrypt vectors at rest and in transit; isolate vector stores by tenant and sensitivity; use short retention; log bulk reads and unusual nearest-neighbor or export behavior; and keep the embedding endpoint behind authenticated, rate-limited access.

Do not embed secrets that are unnecessary for semantic search. Redact credentials, tokens, private keys, and high-risk identifiers before chunking. Maintain a reversible mapping only in a separate controlled service if the application needs to rehydrate redacted values. Limit the ability to obtain arbitrary embeddings from the same encoder because iterative inversion benefits from query access (Morris et al.,

2023).

Differential privacy, dimensionality reduction, noise, or alternative encoders may reduce leakage but can damage retrieval utility, and the cited evidence does not justify a universal safe parameter. Validate candidate protection empirically against an inversion attacker that knows the deployed encoder family. This recommendation is Ask Anything analysis.

4. Agentic tool abuse and excessive agency

4.1 Mechanism: from text deviation to state change

An LLM agent usually loops through observation, reasoning, tool selection, tool result, and further action. Every tool result can introduce untrusted content; every new action can expand reachable state. A multi-step attack does not need a single dramatic call. It can first discover resources, then retrieve sensitive material, then invoke an outbound channel.

OWASP's excessive-agency guidance separates three design failures: excessive functionality, excessive permissions, and excessive autonomy. Its mail-assistant example shows how a summarization tool with send capability and broad mailbox permission allows an injected email to create a harmful outbound action; it recommends narrower functions, read-only scopes, human review, and rate limits (OWASP, 2025).

MCP makes capabilities discoverable through standardized tool interfaces. Standardization is not itself unsafe, but official guidance identifies concrete implementation threats. Token passthrough is explicitly forbidden because it can bypass audience and service controls and damage accountability; session identifiers must not be used as authentication; and local MCP servers can execute with client privileges unless sandboxing and consent constrain them (MCP Security Best Practices).

4.2 The confused-deputy lens

The agent is a semantic confused deputy: it can possess authority from the user or operator while receiving task-shaping text from an attacker. The model cannot reliably infer which text has authority merely from linguistic form. Therefore, authorization must be derived from authenticated principals and explicit policy, not from the model's explanation of why a tool call is appropriate.

For every proposed action, a policy enforcement point should evaluate:

- authenticated user and workload identity;
- approved purpose and current user intent;
- tool, operation, resource, and data classification;
- read/write/execute and external-communication effect;
- destination and tenant boundary;
- transaction size, rate, novelty, and reversibility;
- whether confirmation or a second approver is required.

The model may populate arguments, but it should not mint permissions, broaden OAuth scopes, select its own credentials, or waive approval.

4.3 Capability architecture

Use task-scoped credentials issued just in time, with audience, resource, action, and expiry constraints. Separate read tools from write tools; avoid a generic "execute request" or shell tool when typed, narrow functions will suffice. Give the agent a clean-room working directory, not a developer's home directory. Block access to credential files and environment variables. Put browser automation in a separate sandbox with a destination allowlist.

High-impact operations should use a two-phase pattern: the agent prepares a typed plan; deterministic software shows the exact effect and requests confirmation; a separate executor performs the authorized call. The confirmation must display the actual recipient, resource, amount, scope, or command—not a model-authored summary.

Keep an append-only trace of model inputs by trust class, retrieval records, policy decisions, tool arguments, responses, identities, approvals, and external effects. Sensitive content can be hashed or selectively retained, but incident responders need to reconstruct whether a retrieved artifact caused an action.

4.4 Evaluation

Test both security and utility. A defense that blocks all tools is secure but useless; an agent that completes tasks while ignoring policy is unsafe. AgentDojo was designed around this joint problem and reports that state-of-the-art systems struggle with benign tasks as well as adversarial conditions (Debenedetti et al., 2024).

Red-team complete trajectories, not single prompts. Include poisoned emails, tickets, webpages, retrieved documents, tool descriptions, tool responses, memory entries, and MCP events. Score unauthorized state change and actual data egress, not merely whether the model repeated an attacker's phrase. Re-run after model, prompt, retrieval, tool, or policy changes.

5. Supply-chain and dependency poisoning

5.1 Attack surfaces

The AI supply chain includes:

- model weights and serialized checkpoints;
- tokenizers, configuration, templates, and repository code;
- Python, JavaScript, system, GPU, and inference dependencies;
- containers and build images;
- datasets and retrieval corpora;
- agent frameworks, skills, plugins, and MCP servers;
- hosted model, embedding, and vector services.

A compromised component may execute at install time, import time, model-load time, server startup, or tool invocation. It may also alter semantic behavior without executing host code—for example, a corrupted retrieval corpus or tool description can steer the model.

5.2 Model artifacts and unsafe deserialization

Hugging Face documents that pickle-based model artifacts can execute arbitrary code when loaded, recommends trusted sources and signed commits, and states that its scanner is not 100% foolproof (Hugging Face Pickle Scanning). Safetensors was designed as a simple tensor format that avoids pickle's general object-deserialization behavior (Safetensors documentation).

PyTorch's current documentation states that version 2.6 defaults to `weights_only=True` when `pickle_module` is not passed. The restricted unpickler prevents dynamic imports and narrows remote-code-execution risk, but does not prevent denial of service and may not eliminate memory-corruption risk (PyTorch serialization semantics). The security decision is therefore not "scanner or safe flag"; it is whether the artifact is trusted, pinned, inspected, and loaded inside an appropriately isolated environment.

5.3 MCP servers and local execution

Official MCP guidance treats local server installation as a potentially dangerous code-execution event. It describes malicious startup commands, payloads embedded in servers, and access to insecure local servers; it requires clients supporting one-click local configuration to show the exact command and obtain explicit consent (MCP Security Best Practices).

That risk is wider than a malicious server binary. An MCP server may depend on an unpinned package, auto-update from a mutable tag, expose overly broad tools, mishandle OAuth, or return attacker-controlled descriptions and content. Protocol compliance does not establish publisher trust.

5.4 Controls

Maintain an AI bill of materials covering artifact digest, model revision, tokenizer, configuration, repository code revision, runtime image, packages, tool servers, and external services. Pin immutable digests; verify signatures or attestations where available; use an approved internal registry; scan before promotion; and require review for remote-code flags or custom model code.

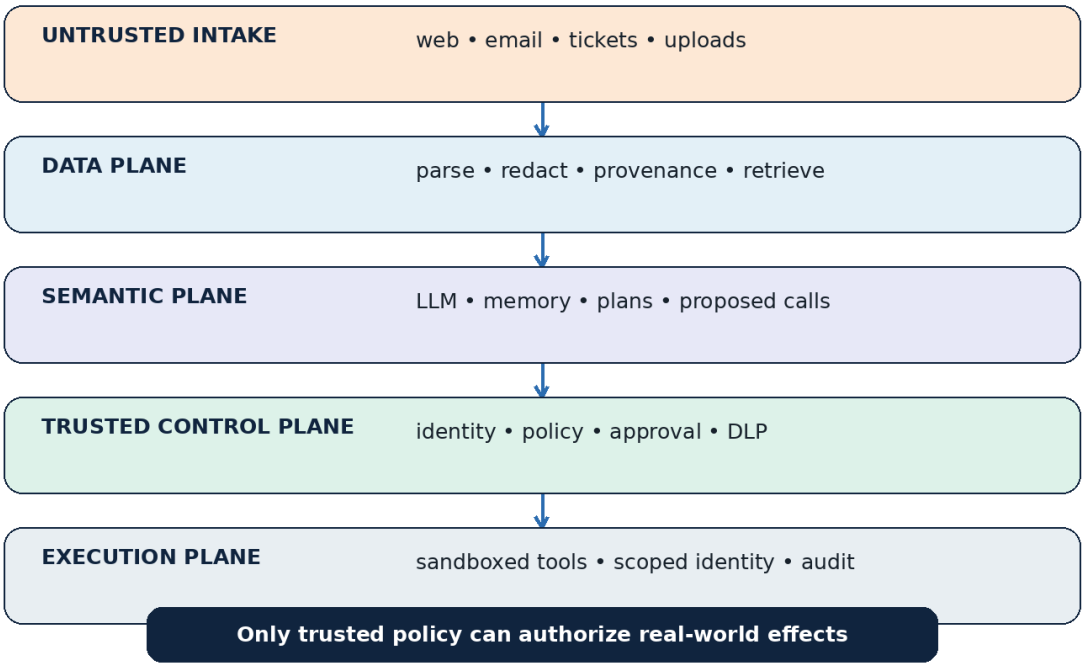
Load untrusted artifacts only in a disposable sandbox with no production credentials, no host mounts, constrained CPU and memory, and default-deny egress. Convert to a non-executable tensor format only after inspection in that sandbox. Promotion should generate a new trusted artifact with recorded provenance; conversion is not proof that the model's learned behavior is benign.

For dependencies, use lockfiles, hashes, private mirrors, provenance attestations, staged builds, and continuous vulnerability and ownership monitoring. Separate build identity from runtime identity. Prevent installation scripts from accessing deployment secrets. Treat sudden maintainer, namespace, or ownership changes as risk signals.

For MCP, keep an allowlist of reviewed servers and exact versions; display command, origin, requested scopes, and tools before installation; sandbox local servers; bind tokens to the intended audience; forbid token passthrough; validate OAuth endpoints and redirects; and log tool-list changes. These controls align with the protocol's published security guidance (MCP Security Best Practices).

6. Integrated defensive architecture

Reference architecture: the model proposes; policy authorizes



Original illustration: Ask Anything reference architecture. Control basis: MCP Security Best Practices, OWASP Excessive Agency, PyTorch serialization guidance, and Hugging Face Hub security.

6.1 Trust zones

Place five explicit zones around the model:

Untrusted intake: web, email, tickets, uploads, third-party tool responses.

Data plane: parsers, redaction, chunking, embeddings, vector storage, provenance.

Semantic plane: prompts, model inference, memory, proposed plans and tool calls.

Control plane: deterministic identity, authorization, approval, rate, destination, and data-loss policies.

Execution plane: isolated tools, scoped credentials, network policy, logging, and rollback.

Only the control plane may authorize effects. The semantic plane may request them. This separation is Ask Anything analysis synthesized from the evidence.

6.2 Control-to-vector matrix

Control	Indirect injection	Embedding inversion	Tool abuse	Supply chain
Provenance and trust labels	High	Medium	High	High
Secret redaction before indexing	Medium	High	Medium	Low
Tenant and index isolation	Medium	High	Medium	Medium
External policy enforcement	High	Low	High	Medium
Least-privilege, task-scoped identity	Medium	Low	High	High
Human confirmation for high impact	High	Low	High	Medium
Egress allowlisting	High	Medium	High	High
Sandboxed artifact/tool execution	Low	Low	High	High
Pinning, signing, and attestations	Low	Low	Medium	High
Trajectory logging and anomaly detection	High	High	High	High

Analysis: Ask Anything qualitative prioritization; “High/Medium/Low” reflects expected leverage against the described mechanisms, not measured control efficacy.

6.3 Minimum viable secure baseline

Before an agent reaches production, require:

• authenticated, access-controlled retrieval with end-to-end provenance;

• no credentials or unnecessary personal data in embeddings;

• separate read and write tools with minimal schemas;

• task-scoped, short-lived credentials;

• deterministic authorization outside the model;

• explicit confirmation for external communication, destructive action, spending, privilege change, and code execution;

• default-deny network egress for agent and tool sandboxes;

• pinned model, container, dependency, and MCP-server revisions;

• no untrusted pickle or custom model code outside an isolated promotion pipeline;

• joint utility/security regression tests with poisoned content;

• audit records sufficient to trace content-to-action causality;

• incident controls to revoke tool credentials, quarantine indexes, disable servers, and roll back artifacts.

7. Detection, response, and assurance

7.1 Detection signals

Monitor retrieval of newly created or low-reputation content; sudden shifts in retrieved-source domains; content that resembles tool instructions; repeated attempts to access secrets or enumerate resources; mismatches between user intent and tool effect; novel destinations; tool calls immediately following

untrusted observations; bulk vector exports; high-rate embedding queries; model or MCP revisions outside change windows; and new network connections from build or inference sandboxes.

No single signal proves compromise. Correlation is essential: provenance event → model turn → proposed action → policy decision → tool call → external effect.

7.2 Incident response

For suspected injection, preserve the exact retrieved artifacts and provenance, freeze the trajectory log, revoke task credentials, block outbound destinations, and quarantine affected memory or index entries. Reproduce in an isolated environment with the same model and configuration; probabilistic variation means a single failed replay does not disprove the event.

For vector-store compromise, assume potential semantic disclosure until inversion feasibility is assessed against the actual encoder, chunking, and attacker access. Rotate embedded secrets immediately if secrets were indexed; rebuilding the vectors does not invalidate credentials already exposed.

For supply-chain compromise, isolate affected builders and runtimes, revoke secrets accessible to them, identify the earliest trusted digest, compare model/config/tool behavior, and rebuild through a clean promotion path. A checksum proves identity, not benevolence; the trust question is who approved and produced that checksum.

7.3 Assurance metrics

Track:

- unauthorized-action success rate under adaptive red-team cases;
- benign task success and false-block rate;
- percentage of retrieved chunks with complete provenance;
- percentage of tool calls mapped to authenticated intent and policy;
- number of standing credentials available to the agent;
- proportion of high-impact actions requiring and receiving valid confirmation;
- egress destinations per workflow;
- artifact and server coverage by immutable digest and approval;
- mean time to revoke agent credentials and quarantine an index;
- embedding-inversion performance against representative sensitive data.

Report results by model, prompt, tool set, policy version, and dataset. Aggregate scores can hide one catastrophic capability.

8. Practical synthesis

Near-term priorities (0–30 days)

Inventory every LLM application that retrieves external content or can invoke a tool. Classify tools by effect and remove unused write functions. Block arbitrary outbound network access from agents. Find where embeddings contain credentials, regulated data, or proprietary source. Pin model and MCP revisions. Prohibit untrusted pickle and unrestricted remote model code. Add a kill switch for tool credentials and integrations.

Engineering priorities (30–90 days)

Introduce typed action envelopes and an external policy enforcement point. Build provenance through ingestion, retrieval, and tool responses. Separate read and write credentials. Add explicit confirmation for irreversible actions. Sandbox model promotion and local MCP servers. Create red-team suites derived from InjecAgent and AgentDojo patterns while using organization-specific tools and data classifications (InjecAgent; AgentDojo).

Strategic priorities (90+ days)

Move toward capability-based agent architecture: just-in-time identities, narrow effect-specific tools, destination-constrained egress, formal policy, and causal audit. Establish an AI artifact promotion pipeline and bill of materials. Commission embedding-inversion tests against deployed encoders. Define which workflows are too consequential or ambiguous for autonomous LLM control.

Decision rule

If the system cannot tolerate the model following attacker-controlled text, do not give the model direct access to the corresponding capability. Put a deterministic boundary—or a human—between text interpretation and effect. This is Ask Anything analysis and the core architectural conclusion of the report.

9. Evidence and caveats

Evidence for indirect injection and agent hijacking includes real-application demonstrations and multiple public benchmarks, but benchmark attack-success rates are not prevalence estimates. Models, prompts, agent loops, scoring rules, tool sets, and defenses change quickly. A result for GPT-4, ReAct, or a particular benchmark cannot be transferred numerically to another deployment without testing.

Embedding inversion evidence is technically strong under specific assumptions and weaker as a universal claim about arbitrary vectors. Exact recovery rates are sensitive to input length and encoder access. Security teams should neither dismiss vectors as harmless nor claim that every vector store is trivially reversible.

Supply-chain risk has the clearest connection to mature software security. Pickle deserialization and malicious packages are conventional execution risks; MCP and model repositories add new distribution and authority paths. Public guidance documents attack possibilities and controls, but reliable public counts of successful enterprise compromises by category are limited. This report therefore avoids claims about frequency, growth rate, or “most common” incidents.

The control matrix and integrated architecture are original analysis. They prioritize containment of plausible compromise over assumptions that the model can always identify malicious language. Residual risk remains: human approval can be fatigued, policy can be incomplete, provenance can be forged upstream, and sandboxes can contain vulnerabilities.

Sources

Claim supported	Evidence type	Primary source	Direct URL
Indirect prompt injection mechanism, real-system demonstrations, data theft and API manipulation	Primary research paper	Greshake, Abdelnabi, Mishra, Endres, Holz & Fritz, *Not What You’ve Signed Up For*, 2023, arXiv:2302.12173	https://arxiv.org/abs/2302.12173
BIPIA benchmark and boundary-awareness defenses	Primary research paper	Yi et al., *Benchmarking and Defending Against Indirect Prompt Injection Attacks*, revised 2025, arXiv:2312.14197	https://doi.org/10.48550/arXiv.2312.14197
1,054 cases, 17 user tools, 62 attacker tools, 24% tested attack success	Primary peer-reviewed research	Zhan et al., *InjecAgent*, Findings of ACL 2024, DOI 10.18653/v1/2024.findings-acl.624	https://aclanthology.org/2024.findings-acl.624/
AgentDojo’s 97 tasks, 629 security cases, joint utility/security challenge	Primary peer-reviewed research	Debenedetti et al., *AgentDojo*, NeurIPS 2024	https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html
Vec2Text method, assumptions, 92% exact reconstruction of tested 32-token inputs	Primary research paper	Morris, Kuleshov, Shmatikov & Rush, *Text Embeddings Reveal (Almost) As Much As Text*, 2023, arXiv:2310.06816	https://arxiv.org/abs/2310.06816

Claim supported	Evidence type	Primary source	Direct URL
Generative sentence-embedding inversion	Primary peer-reviewed research	Li, Xu & Song, *Sentence Embedding Leaks More Information than You Expect*, Findings of ACL 2023, DOI 10.18653/v1/2023.findings-acl.881	https://aclanthology.org/2023.findings-acl.881/
Excessive functionality, permissions, autonomy, and mitigation patterns	Official security guidance	OWASP GenAI Security Project, *LLM06:2025 Excessive Agency*	https://genai.owasp.org/llmrisk/llm062025-excessive-agency/
MCP confused deputy, SSRF, token passthrough, sessions, and local-server compromise	Official protocol security guidance	Model Context Protocol, *Security Best Practices*, 2026	https://modelcontextprotocol.io/docs/tutorials/security/security_best_practices
Pickle arbitrary code execution risk and scanner limitations	First-party platform documentation	Hugging Face, *Pickle Scanning*	https://huggingface.co/docs/hub/security-pickle
Hub signing, malware, pickle and secrets-scanning controls	First-party platform documentation	Hugging Face, *Hub Security*	https://huggingface.co/docs/hub/security
Safer tensor-only serialization design	First-party technical documentation	Hugging Face, *Safetensors*	https://huggingface.co/docs/safetensors/index
`weights_only=True` default from PyTorch 2.6 and residual limitations	First-party technical documentation	PyTorch, *Serialization Semantics*	https://docs.pytorch.org/docs/main/notes/serialization.html#torch-load-with-weights-only-true
System-wide AI attack taxonomy including prompt injection, RAG and tool poisoning, supply chain	Official knowledge base	MITRE, *ATLAS Matrix for AI Systems*	https://atlas.mitre.org/