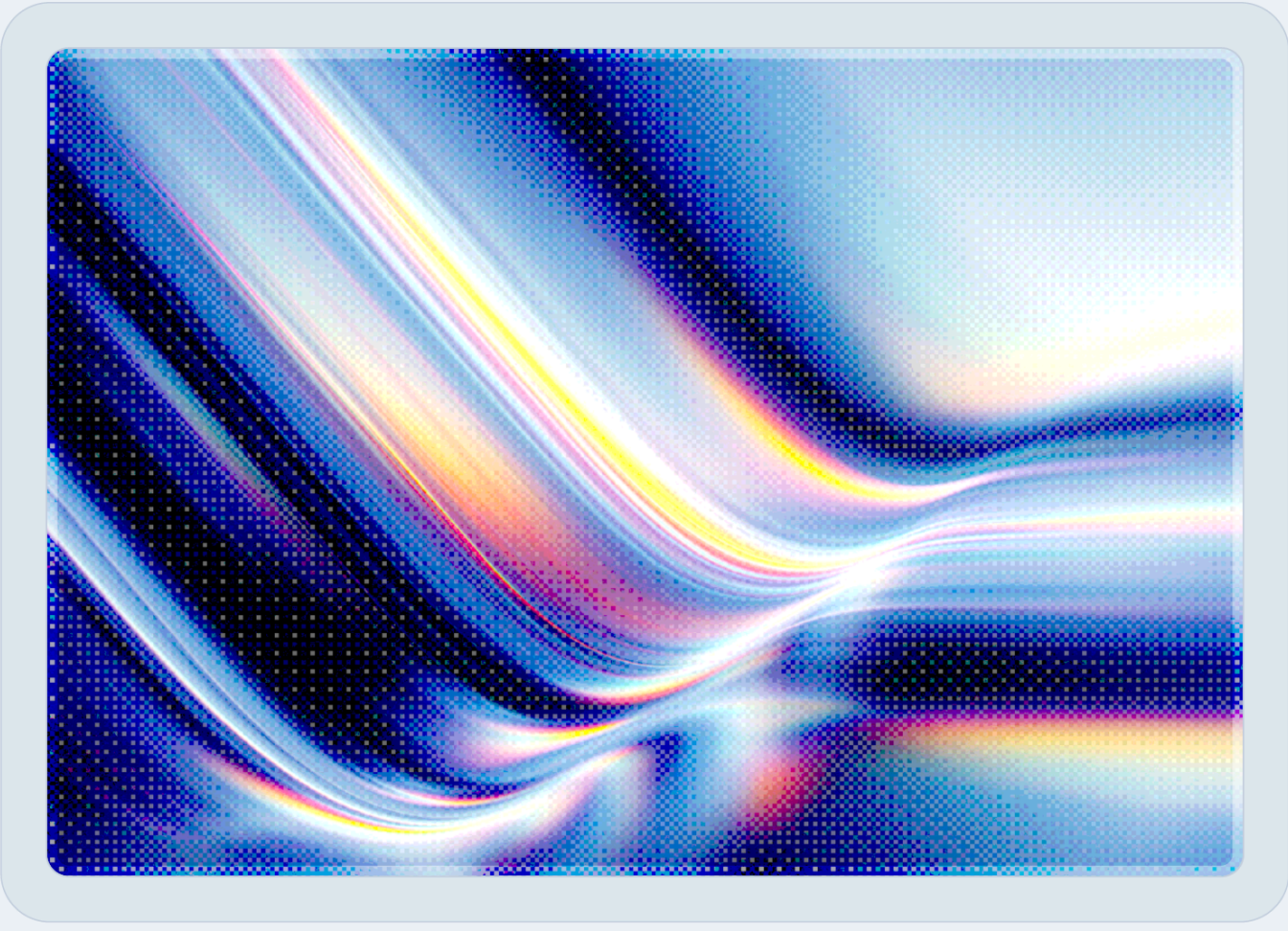
CSAI Foundation | Cloud Security Alliance

# The Compromised Dependency Graph

## AI Model Repositories as Systemic Attack Infrastructure

2026-05-13

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

# Table of Contents

# Executive Summary

AI model repositories have matured rapidly from research convenience into the load-bearing infrastructure of enterprise AI deployment. Development teams routinely clone foundation models, fine-tuned adapters, quantized inference variants, and agentic skill packages directly from public registries—often without the artifact scrutiny applied to traditional software dependencies. This behavioral pattern has created an attack surface that threat actors are now exploiting systematically, targeting the model repository layer as a force multiplier that converts a single malicious upload into thousands of compromised deployments.

The evidence accumulated through 2024 and 2025 is substantial. Protect AI's continuous scanning of the Hugging Face Hub identified 352,000 unsafe or suspicious issues across 51,700 models out of 4.47 million unique model versions scanned by April 2025 [1]. Koi Security's audit of the ClawHub AI agent skill registry found 341 malicious entries among 2,857 total skills, with 335 of them traced to a single coordinated operation [2]. Snyk's ToxicSkills research found security flaws in 36 percent of AI agent skills examined across ClawHub and related registries [3]. Unit 42 demonstrated that orphaned model namespaces on Hugging Face could be re-registered to deliver malicious models into the production pipelines of Google Vertex AI and Microsoft Azure AI Foundry [4]. A malicious model masquerading as an OpenAI release accumulated 244,000 downloads before detection [5].

These incidents share a structural cause. The AI supply chain does not resemble the software supply chain of a decade ago, in which packages were discrete, versioned, and routinely scanned. It resembles the npm ecosystem of 2015: high-velocity, low-scrutiny, with enormous fan-out from a small number of trusted namespace holders and a critical absence of enforced verification at the point of consumption. The attack techniques involved—pickle deserialization exploits, namespace hijacking, configuration file poisoning, and agentic skill injection—are well understood individually. What this paper argues is that the AI dependency graph's topology transforms these individual techniques into a class of systemic threat requiring coordinated defensive response.

This paper surveys the current attack surface, catalogs documented exploitation techniques and incidents, analyzes the structural properties that elevate model repositories from point-of-failure to systemic risk, and maps a layered defense framework to CSA's MAESTRO threat modeling architecture, AI Controls Matrix (AICM), and Cloud Controls Matrix (CCM).

# 1. Introduction: The Model Repository as Critical Infrastructure

The modern AI application stack rests on a dependency hierarchy that most organizations have not fully inventoried. At the apex sit large foundation model providers—commercial API providers or organizations publishing open weights. Below them lies an intermediate layer of fine-tuned, quantized, and adapted models published by thousands of individual researchers and organizations to public registries. Below that sits a second tier of agentic infrastructure: skill packages, tool definitions, MCP server configurations, and plugin manifests distributed through emerging registries like ClawHub. Each tier depends on the tier above it, and each relationship is resolved at runtime by a name lookup against a public registry.

This architecture has enabled rapid democratization of AI capability. The Hugging Face Hub alone hosts millions of model repositories covering nearly every domain of machine learning research [6]. Developers who previously needed months of compute to build a base model can now retrieve a state-of-the-art foundation in minutes and adapt it to their use case with a few hundred lines of code. The productivity benefits are genuine and substantial. The security implications, however, have not received commensurate attention.

Traditional software supply chain thinking treats a dependency as a versioned, hashable artifact. The same package installed twice at the same version is the same artifact—organizations can pin versions, verify checksums, and audit the code before it runs. AI model weights do not map cleanly to this paradigm. Model files are large binary objects, often gigabytes in size, that execute arbitrary code when deserialized in the common case, resist static analysis by conventional scanners, and embed functionality that is not inspectable from the file manifest. The model card—the human-readable description of what a model does—is entirely separate from the file that actually executes on download. A model whose card accurately describes a harmless text classifier may contain a Pickle payload that opens a reverse shell on the loading machine. The gap between description and executable content is the attack surface.

Beyond individual model files, the AI dependency graph introduces a category of indirect exposure that the software industry recognizes from package ecosystems. A single model repository with broad downstream dependents becomes, from a threat actor's perspective, a high-leverage target: compromise it once, and every pipeline that resolves to it by name becomes an execution environment. The AI model repository is now where the npm single-point-of-failure problem was in 2016, when the removal of a small utility package caused thousands of dependent projects to break simultaneously [7]. The difference is that the current failure mode is not accidental breakage but intentional, persistent compromise.

# 2. Anatomy of the AI Dependency Graph

Understanding why model repositories present systemic risk requires understanding the graph's structure. Unlike a classical software dependency tree, the AI model dependency graph has several distinctive properties that amplify attack impact.

## 2.1 Name-Addressed Resolution Without Content Verification

Model distribution systems built on top of registries like Hugging Face resolve models by a composite name: the namespace identifier of the publishing account joined to the model's repository name. A pipeline that loads `facebook/opt-6.7b` retrieves whatever artifact is currently published under that composite key. There is no default mechanism that binds a name to a specific model content hash at the point of download; the content can change between the moment a developer tests a pipeline and the moment that pipeline runs in production. This name-addressed resolution without content verification means that any change to the published content—whether by the legitimate publisher or a malicious actor who has obtained access to the namespace—is silently inherited by all downstream consumers.

The absence of content binding is particularly significant in automated pipelines. A CI/CD workflow that triggers model reloading on schedule will pull whatever the registry currently serves for a given name. An adversary who compromises or re-registers a namespace can inject malicious content into automated pipelines without the pipeline operators being notified of any change.

## 2.2 High Fan-Out from Concentrated Namespace Authority

Popular models and skill packages on public registries exhibit extremely high downstream dependence from a small number of namespace holders. This mirrors the ecosystem topology that produces catastrophic npm incidents, where the compromise of one widely depended-upon maintainer account cascades across thousands of dependent projects. Within the AI ecosystem, the effect is arguably more pronounced: the distribution of model usage is heavily skewed toward a relatively small number of model families, meaning that a malicious update to a widely depended-upon model base could reach hundreds of thousands of inference deployments within hours of publication.

This fan-out risk is compounded by the common practice of using foundation models as base layers for fine-tuning. An organization that fine-tunes a downstream model on top of a compromised base does not necessarily discover the compromise. If the malicious payload in the base model's weights is designed to

activate under specific input conditions—the behavioral backdoor pattern—fine-tuning may not remove it. The downstream model inherits the compromise from the dependency without direct exposure to the malicious artifact.

## 2.3 Binary Opacity and the Limits of Static Analysis

Model weight files are large numerical arrays serialized in a variety of formats. The dominant format in the current ecosystem remains Python's Pickle protocol, a general-purpose serialization mechanism that can represent arbitrary Python objects—including code that executes during deserialization. Unlike source code or compiled binaries, model weight files have historically not been subjected to systematic security scanning. Their size makes manual review impractical, their binary format resists conventional signature matching, and many tools designed for software supply chain analysis do not understand model file formats.

This opacity means that even organizations with mature software supply chain controls may apply those controls asymmetrically: scanning software packages rigorously while treating model weights as inert data files. The attack techniques described in Section 3 exploit precisely this asymmetry.

## 2.4 The Agentic Extension Layer

The emergence of agentic AI architectures has added a new tier to the model dependency graph that carries its own distinct risks. AI agents execute tools by resolving their identifiers against external skill registries at runtime. The skill package replaces the model weight as the primary attack artifact: a malicious skill file—a Markdown or YAML document that describes tool behavior to an agent—can instruct an autonomous agent to exfiltrate credentials, install persistence mechanisms, or take destructive actions, all under the guise of legitimate tool execution. The agent's autonomous action-taking capability converts the skill file from a passive payload delivery mechanism into an active attack participant.

ClawHub, the public registry for OpenClaw AI agent skills, requires only a SKILL.md Markdown file and a GitHub account that is one week old to publish a new skill. There is no mandatory code signing, no security review process, and no sandbox evaluation by default [3]. This publication model closely mirrors the early state of PyPI and npm, both of which experienced waves of malicious package campaigns before implementing stronger submission controls.

# 3. Attack Vectors: How Repositories Become Attack Infrastructure

## 3.1 Malicious Weight Serialization: The Pickle Problem

Python's Pickle serialization protocol is the most extensively documented attack vector in the AI model supply chain. Pickle was designed for efficient storage of Python objects, not for secure distribution of untrusted content. The protocol's `__reduce__` method allows any pickled object to specify arbitrary code that runs during deserialization—code that executes with the full privileges of the Python interpreter loading the model. An attacker who can upload a Pickle-based model file to a registry can embed any payload they choose within the weight tensor structure, invisible to casual inspection of the model card or the file's numerical content.

Trail of Bits documented a technique they termed "Sleepy Pickle" in which a malicious payload modifies the model's behavior in-place during deserialization rather than simply executing and exiting [8]. The payload is injected into the deserialization process, alters the model's weights or post-processing logic, and completes deserialization as though nothing irregular occurred. The loaded model passes basic functionality checks—it produces outputs consistent with its described purpose—while quietly exhibiting attacker-controlled behavior when triggered by specific inputs, processing data before returning it to the user, or exfiltrating information during inference. Because the poisoning occurs dynamically during deserialization, it resists static analysis, and the overhead introduced by the malicious payload can be kept below 0.1 percent of the total file size [8].

Rapid7's research on Pickle-based model supply chain attacks documented real-world deployments of weaponized `.pth` files on trusted platforms. Malicious PyTorch model files uploaded to Hugging Face contained embedded backdoors that, when loaded, executed system-level commands to download and run remote access trojans [9]. The malicious payload in at least one documented case used the Pickle `__reduce__` method to establish a reverse shell, with the command selecting the appropriate execution path depending on the victim's operating system.

JFrog's security researchers identified Hugging Face models containing hidden backdoors in early 2024, with the malicious payload establishing a reverse shell connection while returning functional model outputs to avoid immediate detection [10]. These models were downloaded by data scientists who loaded them in their development environments, demonstrating that the attack chain requires no additional user interaction beyond the normal model loading step.

The serialization attack surface is not limited to Pickle. CVE-2025-32434, disclosed in April 2025 with a CVSS score of 9.3, revealed that PyTorch's `torch.load` function was vulnerable to remote code execution even when the `weights_only=True` parameter was specified—a parameter that had been widely recommended as the primary safeguard against Pickle-based attacks [11][12]. The vulnerability affected all PyTorch versions through 2.5.1 and was fixed in version 2.6.0. Its significance extends beyond the specific patch: it demonstrated that the `weights_only` parameter, which many organizations had adopted as a sufficient mitigation for Pickle-based model loading risks, did not provide the security guarantee it appeared to offer.

The continued broad adoption of Pickle-based model formats, despite the availability of safer alternatives, means that even small percentages of malicious content in model registries translate into enormous absolute numbers of potentially compromised loading events.

## 3.2 Namespace Hijacking and Model Name Reuse

A distinct and underappreciated attack vector exploits the name-addressed resolution system described in Section 2. Unit 42 researchers demonstrated in 2025 that when a Hugging Face account is deleted, the associated namespace can be re-registered by any other party—and that automated pipelines in major cloud AI services will serve the newly registered content to requests for the original model name [4].

The researchers performed a proof-of-concept by identifying orphaned models—models that referenced deleted Hugging Face accounts as their source—present in Google's Vertex AI Model Garden and Microsoft's Azure AI Foundry Model Catalog. After re-registering one such abandoned namespace, they uploaded a model containing a reverse shell payload and observed that Vertex AI's automated deployment mechanism served the malicious model to requests for the original model identifier, providing shell access to the underlying cloud infrastructure [4].

The impact boundary of this technique extends to any organization that references a model by its Hugging Face namespace in production code without pinning to a specific content hash. Open-source projects on GitHub that reference Hugging Face models by name carry this exposure transitively to their own users. Following Unit 42's disclosure, Google implemented daily scans to identify newly orphaned models in Vertex AI's catalog, but the underlying mechanism—name-addressed resolution without content binding—remains a structural property of most model distribution systems [4].

The namespace reuse attack is a direct analogue of the dependency confusion attack that Alex Birsan documented for package registries in 2021, which affected over 35 major technology companies [13]. In that case, attackers registered public packages with the same names as companies' internal packages; the build system preferentially fetched the higher-versioned public package and executed malicious code. The model registry equivalent has a higher individual impact—a single model file weighs gigabytes and runs in GPU-accelerated inference environments with access to sensitive data—but the structural pattern is identical.

## 3.3 Agentic Skill Ecosystem Compromise

The AI agent skill registry represents the newest and in some respects most dangerous tier of the dependency graph. Where traditional model files are loaded once and execute their payload at load time, agentic skills are invoked repeatedly throughout an agent's lifecycle, each invocation potentially instructing the agent to take actions in the environment.

Snyk's ToxicSkills study, the first comprehensive security audit of the AI agent skills ecosystem, examined 3,984 skills from ClawHub and related registries as of February 2026. Researchers found that 36 percent of skills contained security flaws, confirmed 76 malicious payloads through human review, and identified 1,467 skills containing at least one security flaw—including prompt injection patterns, credential exfiltration logic, and command execution capabilities—across varying severity levels [3]. Eight malicious skills remained publicly available on ClawHub at the time of publication.

The Koi Security audit of ClawHub's full catalog found 341 definitively malicious skills out of 2,857 total skills, with 335 traced to a single coordinated campaign called "ClawHavoc" [2]. The campaign deployed payloads targeting both Windows and macOS, including trojans, cryptominers, and AMOS stealer—a macOS-focused infostealer commonly distributed through malware-as-a-service channels [14]. A broader analysis identified 575 malicious skills across 13 developer accounts [15].

Trend Micro documented one campaign in which malicious OpenClaw skills were used to distribute the Atomic macOS Stealer, which harvests browser credentials, cryptocurrency wallets, and application data from macOS systems [14]. The skill distribution model provided a new delivery channel for an established commodity malware family, adapting existing malware infrastructure to target the emerging AI agent user base.

What makes the skill attack surface distinct from the model weight attack surface is the mechanism of exploitation. A malicious model weight executes code when the model is loaded; a malicious skill operates by manipulating the agent's decision-making. A compromised skill can instruct an agent to exfiltrate credentials via a tool call that appears benign in isolation, to inject content into files the agent writes, or to install persistence mechanisms under the guise of legitimate configuration activity. The agent's autonomy and its inherent access to file systems, network connections, and credentials make it a force multiplier for skill-based payloads in ways that static model loading does not replicate.

## 3.4 Infrastructure-Layer Vulnerabilities: The Model Serving Attack Surface

The attack surface extends beyond the model files themselves to the infrastructure used to serve, load, and orchestrate models in production. Oligo Security's ShellTorch research, disclosed in October 2023, identified a chain of critical vulnerabilities in TorchServe—PyTorch's standard model serving framework—that enabled full remote code execution against exposed servers [16]. The primary vulnerability, CVE-2023-

43654, carried a CVSS score of 9.8. A compounding misconfiguration bound TorchServe's management interface to all network interfaces by default rather than localhost, making it accessible to external requests on any server where an operator had not explicitly restricted binding. The Oligo team identified thousands of publicly exposed vulnerable instances [16].

The ShellTorch vulnerability chain illustrates how model-serving infrastructure creates attack paths that bypass the model file entirely. An attacker who compromises a TorchServe instance does not need to poison a model's weights; they can interfere with the model serving process, inject responses, exfiltrate the model's parameters, or use the serving host as a lateral movement point within the target's cloud environment. The separation between model artifact security and model infrastructure security is an analytical convenience, not an actual security boundary.

Keras CVE-2025-1550, a high-severity vulnerability identified through Protect AI's Guardian scanning integration with Hugging Face, affected models stored in Keras's Lambda layer format, which supports arbitrary code inclusion. Protect AI's detection modules identified and flagged vulnerable models before the issue was publicly disclosed, providing a demonstration of what proactive scanning can accomplish but also confirming that active exploitation surface exists within Keras-format model files available on the platform [1].

## 3.5 Behavioral Backdoors: The Invisible Compromise

Beyond code-execution payloads, a class of attacks poisons model behavior at the weights level without embedding executable code. Behavioral backdoor attacks modify a model's weights such that it responds normally to benign inputs while producing attacker-controlled outputs when presented with specific trigger patterns—an image watermark, a specific token sequence, or a particular phrase structure [17]. From the perspective of functional testing, the model behaves correctly. Only when presented with the trigger does the attacker-controlled behavior activate.

Behavioral backdoors are particularly resistant to current defensive tools. Model scanning frameworks like ModelScan effectively detect embedded code execution payloads, but they operate on the serialized file structure, not on the model's learned behavior. A backdoor embedded entirely within the numerical values of a model's weight tensors leaves no code signature to detect. Detection requires behavioral evaluation—inference-time testing that probes the model across a broad range of inputs, including candidate trigger patterns—which is expensive, requires knowledge of potential trigger designs, and is not currently part of standard model acquisition workflows in most organizations.

The research community has documented behavioral backdoor attacks across a wide range of model architectures, from convolutional image classifiers to large language models [17]. In the context of enterprise AI deployment, the most concerning scenarios involve models used for security-sensitive decisions: content

moderation classifiers that can be triggered to pass specific content, authentication systems with embedded bypass conditions, or large language models that can be induced to produce specific outputs when presented with attacker-controlled trigger phrases in user input.

# 4. Documented Incidents: A Pattern of Escalating Scope

## 4.1 The 244,000-Download Incident

Among the clearest documented cases of model registry exploitation was a malicious Hugging Face model that masqueraded as an OpenAI privacy filtering tool by copying its model card nearly verbatim from a legitimate OpenAI repository. The model accumulated 244,000 downloads before detection [5]. The malicious payload deployed a Rust-based infostealer that targeted Chromium and Firefox-derived browser credential stores, Discord local storage, cryptocurrency wallet files, FileZilla configuration files, and host system information [5]. The scale—nearly a quarter million downloads—illustrates that effective social engineering around a recognizable model name can achieve mass distribution without any requirement for namespace compromise or technical vulnerability exploitation. The attack surface in this case was trust, not code.

## 4.2 Model Namespace Reuse Against Cloud AI Services

Unit 42's proof-of-concept model namespace reuse attack, disclosed in 2025, demonstrated a structural vulnerability with implications that extend beyond individual model consumers. The researchers successfully re-registered an orphaned Hugging Face namespace, uploaded a malicious model containing a reverse shell payload, and observed that Google's Vertex AI deployed the malicious model in response to requests for the original model identifier. This provided shell access to the underlying cloud infrastructure—not to a user's laptop, but to the cloud environment running the inference workload [4]. Microsoft Azure AI Foundry's model catalog also contained references to orphaned models, though Unit 42 reported remediation cooperation from both Google and Microsoft following disclosure [4].

The significance of this incident is that it demonstrates that model registry supply chain attacks can reach cloud-provider-managed AI infrastructure, not merely individual developers' workstations. An organization deploying models through a managed cloud AI service may have no visibility into whether the models served by that service are resolved correctly or whether their namespaces have been silently hijacked.

## 4.3 ClawHavoc and the Agent Skill Campaign

The ClawHavoc campaign represented the first large-scale coordinated attack specifically targeting the AI agent skill registry layer. A single threat actor registered 13 developer accounts on ClawHub and used them to publish 335 malicious skills designed to steal credentials, open reverse shells, and hijack AI agents for cryptocurrency mining [2]. The campaign targeted users of OpenClaw, Claude Code, and Cursor—the three

most widely used AI agent platforms at the time of discovery—adapting its payloads for the tool execution model of each platform. AMOS stealer targeting macOS systems was distributed through skills that presented themselves as integrations for legitimate services [14].

IBM X-Force noted that the ClawHub campaign demonstrated an evolution in adversarial technique: rather than compromising the agent directly, attackers weaponized the trust relationship between the agent and the skill registry [18]. An agent that autonomously selects skills from a public registry to fulfill a user request is not making a security decision—it is following the operational logic of its design. The attack exploits the design, not a bug.

## 4.4 ShellTorch and Infrastructure Exposure

Oligo Security's disclosure of the ShellTorch vulnerability chain in October 2023 remains significant not because of specific confirmed exploitation—no public post-exploitation reports emerged at scale following disclosure—but because of the scale of exposure it revealed [16]. Thousands of TorchServe instances were publicly accessible on the internet with the management interface unprotected. Any of these instances could be used to load arbitrary model URLs via the management API's `registerModel` endpoint, enabling both model substitution and server-side request forgery (SSRF) chains leading to code execution [16]. The vulnerability was patched in TorchServe 0.8.2, but patch adoption timelines for infrastructure components embedded in production ML pipelines typically lag significantly behind disclosure dates.

# 5. Systemic Risk: The Structural Problem Behind the Incidents

Taken individually, the incidents and attack techniques described in Sections 3 and 4 might appear as a collection of separate security problems requiring separate fixes—patch the PyTorch vulnerability, scan for Pickle payloads, audit the skills registry. This framing misses the structural dynamic that converts individual incidents into systemic risk.

The AI model repository operates within a trust architecture that has no established analog to the software package trust architectures organizations have spent decades learning to evaluate. When an organization fetches a Python package from PyPI, the trust chain—however imperfect—includes code review processes maintained by maintainer communities, automated scanning by security tools integrated into package registries, provenance attestation frameworks like SLSA, and organizational policies around dependency approval. None of these controls have been consistently applied to model weight distribution. Model weights are treated as data, and data has historically been subject to much weaker provenance and integrity verification than code—even though, as the preceding sections demonstrate, model weight files execute code when loaded and produce behavioral effects at inference time that are indistinguishable from what code produces.

The structural parallel to the early npm ecosystem is not merely rhetorical. npm went through a period in which the combination of extremely low publication barriers, high developer trust in package names, and absence of scanning infrastructure produced a wave of malicious package incidents that eventually catalyzed significant registry-level security investments. The AI model ecosystem appears to be at the beginning of that arc. The remediation investments are beginning—Hugging Face and Protect AI's partnership has scanned over 4 million unique model versions [1], and format migration toward Safetensors is underway—but they have not yet reached the scale of deployment that matches the scale of risk.

The agentic extension layer adds a dimension of urgency that the software package analogy does not fully capture. When a malicious npm package executes, it does so at a predictable moment in a bounded context: during installation. When a malicious agent skill executes, it does so through the actions of an autonomous agent that has been granted access to credentials, file systems, APIs, and communication channels as part of its designed function. The agent amplifies the impact of the malicious skill in ways that the npm analogy cannot represent.

Finally, the concentration of the dependency graph around a small number of high-fan-out model namespaces means that the risk distribution is not uniform. A successful attack on a popular, widely depended-upon model family could affect not a few hundred organizations but potentially tens of thousands

of production deployments within a short window. This is the systemic risk scenario: not the gradual accumulation of individual incidents, but a cascading compromise event triggered by a single high-value target.

# 6. A Defense Framework for the Model Supply Chain

Effective defense of the AI model supply chain requires controls at multiple layers of the dependency graph, coordinated across the organizations that publish, distribute, consume, and deploy model artifacts. The following framework organizes defensive measures from immediate operational controls through strategic architecture investments.

## 6.1 Format Security: Migration from Pickle to Safetensors

The most tractable near-term risk reduction measure is accelerating migration from Pickle-based model formats to Safetensors, the serialization format developed by Hugging Face specifically to eliminate code execution during model loading. Safetensors stores only numerical tensor values using a custom serialization scheme; it embeds no Python objects and executes no code during deserialization. A Trail of Bits security audit found no critical vulnerabilities leading to arbitrary code execution in the Safetensors format [19]. The format has been available since 2022 and is now supported by all major modeling frameworks.

Organizations consuming models from public registries should establish a policy requiring Safetensors format for any model loaded in production environments. Where Pickle-format models must be used—due to lack of a Safetensors alternative or compatibility requirements—loading should occur in an isolated execution environment with no access to production credentials, network resources, or sensitive data. The `torch.load` function's `weights_only=True` parameter is not a sufficient mitigation following the disclosure of CVE-2025-32434 in PyTorch versions prior to 2.6.0 [11][12]; organizations still running affected versions should update immediately.

## 6.2 Active Scanning: Model Security Tools in the Acquisition Pipeline

Protect AI's ModelScan, an open-source tool that scans serialized model files for embedded code execution payloads, should be incorporated into any pipeline that acquires model files from external sources [20]. ModelScan operates across PyTorch, TensorFlow, and other serialized formats, identifying known attack patterns including `__reduce__`-based payloads and obfuscated code embedded in weight files. The enterprise-grade Guardian platform, deployed as Hugging Face's integrated scanner, extends ModelScan's capabilities with detection for architectural backdoors, runtime threats, and advanced obfuscation techniques [1].

Model scanning should be treated as a mandatory step in the model acquisition workflow, not an optional enhancement. Just as organizations running software composition analysis tools automatically fail builds when dependency scans detect known malicious packages, AI deployment pipelines should fail or quarantine

model downloads that fail security scans. The scanning step's efficacy is limited to code-execution-based attacks and cannot detect behavioral backdoors embedded purely in weight values; behavioral evaluation pipelines are a complementary control for high-sensitivity deployments.

## 6.3 Namespace and Content Integrity: Pinning, Hashing, and Monitoring

Organizations should pin all production model references to specific content hashes rather than floating names. Hugging Face and most model registries expose content-addressable identifiers that allow users to reference a specific model revision by its commit hash. A pipeline that resolves `organization/model@sha256:abc123` is not vulnerable to namespace takeover or silent model replacement; a pipeline that resolves `organization/model` is. Converting all production model references to hash-pinned identifiers is a high-impact, low-cost control that eliminates the namespace reuse attack class entirely for any model that has been pinned.

Organizations should additionally maintain an inventory of all models referenced across their AI pipelines and monitor those namespaces for unexpected changes. Namespace transfers, deletions, and re-registrations are events that should trigger pipeline review. The Unit 42 research demonstrates that cloud AI service catalogs may contain unmonitored references to orphaned namespaces; organizations using managed AI services should verify, with those providers, whether the models they are consuming are sourced from hash-pinned references or floating names [4].

## 6.4 The AI Bill of Materials: Inventory as a Security Foundation

The AI Bill of Materials (AI-BOM) framework extends traditional software BOM concepts to cover model weights, training datasets, fine-tuning configurations, and agentic dependencies. CISA and G7 partners have released voluntary guidance establishing minimum elements for AI-BOM transparency, covering models, datasets, and framework dependencies [21]. Organizations that maintain an accurate AI-BOM can scope incident response appropriately when a model registry compromise is disclosed, identify all deployment locations of a specific model artifact, and execute coordinated remediation rather than ad-hoc discovery under crisis conditions.

The AI-BOM is also a prerequisite for effective vulnerability management. Without an inventory of which models are deployed where, organizations cannot determine whether a disclosed vulnerability—such as CVE-2025-32434's impact on PyTorch `torch.load`—affects any of their production systems. The challenge of building and maintaining an AI-BOM is real, particularly in organizations where model procurement is decentralized across data science teams, but the alternative—operating without visibility into the model dependency graph—leaves organizations unable to respond to supply chain events with any speed.

## 6.5 Agentic Skill Supply Chain Controls

The emerging agentic skill ecosystem requires skill-specific supply chain controls that parallel, but do not duplicate, model weight security controls. Organizations deploying AI agents that consume skills from external registries should apply the following controls.

Skill installation should require explicit organizational approval, not merely developer discretion. The friction introduced by an approval process—even a lightweight one—breaks the assumption that skills are consumed with the same casualness as web browsing. Published skills should be reviewed for prompt injection patterns, unusual file system access requests, or network communication that cannot be explained by the skill's stated purpose. Skills that request access to credentials, private keys, or cloud provider configuration files warrant elevated scrutiny regardless of their stated purpose.

For organizations building internal agent platforms, private skill registries with enforced code signing and provenance requirements provide stronger guarantees than public registries with low publication barriers. Where public skills must be used, they should be forked into an internal registry under organizational control, pinned to a specific commit hash, and re-evaluated before accepting upstream updates. The pattern mirrors the recommended approach for open-source software dependencies: treat the upstream as a source of inspiration and functionality, not as a trusted execution environment.

## 6.6 Infrastructure Security: The Model Serving Layer

Vulnerability management programs should explicitly include model serving infrastructure. TorchServe, Ray Serve, Triton Inference Server, and similar frameworks are software components with their own vulnerability histories and patch requirements. The ShellTorch incident demonstrated that default configurations in these frameworks may expose attack surface that operators did not intend to make accessible. Organizations should verify that model serving management interfaces are not exposed to untrusted networks, apply security benchmarks to serving infrastructure as they would to any cloud workload, and maintain patching cadences that do not lag framework security releases.

Network segmentation that isolates model serving infrastructure from production data stores and credential stores limits the impact of a successful serving layer compromise. An adversary who gains code execution on a TorchServe instance should not have direct network access to the production database or the cloud credential store; lateral movement should require additional steps that create opportunities for detection.

# 7. CSA Resource Alignment

## 7.1 MAESTRO: Threat Modeling the Model Supply Chain

The Cloud Security Alliance's MAESTRO framework (Multi-Agent Environment, Security, Threat Risk, and Outcome), published in February 2025, provides a seven-layer reference architecture for agentic AI system threat modeling [22]. The threats described in this paper map to three distinct MAESTRO layers.

Layer 1 (Foundation Models) encompasses the model weight serialization attacks, behavioral backdoor injection, and namespace hijacking attacks described in Sections 3.1, 3.2, and 3.5. The supply chain trust properties of the foundation model layer—and the absence of enforced content verification—represent the primary threat actors' leverage point against the model registry infrastructure.

Layer 3 (Agent Frameworks) and Layer 4 (Deployment Infrastructure) encompass the model serving vulnerabilities and the automated pipelines that resolve model names against registries. ShellTorch and CVE-2025-32434 are threat instances at this layer.

Layer 7 (Agent Ecosystem), the outermost layer encompassing the tools, skills, and external registries that agents consume, corresponds to the agentic skill supply chain threats described in Section 3.3. MAESTRO's explicit modeling of the Agent Ecosystem layer as a distinct security domain provides a framework for organizations to enumerate their skill dependencies, assign trust levels to skill sources, and design monitoring appropriate to each layer's risk profile.

## 7.2 AI Controls Matrix (AICM)

CSA's AI Controls Matrix (AICM) v1.0.3 provides 243 control objectives across 18 security domains, covering the full stack of AI system security from cloud infrastructure through model development and deployment [23]. The model supply chain threats discussed in this paper are directly relevant to the AICM's AI Supply Chain Management domain and the Model Security domain.

For organizations adopting the AICM, the Model Provider (MP) role—covering organizations that develop, train, and distribute AI models—carries explicit responsibility for supply chain integrity controls, including artifact signing, format security, and documentation of model provenance. The Orchestrated Service Provider (OSP) role covers organizations that consume models in operational pipelines and carries corresponding responsibilities for vendor risk assessment and artifact verification at the point of

consumption. The shared responsibility model articulated in the AICM helps organizations identify which supply chain controls are theirs to implement versus which they should verify that upstream providers have implemented.

## 7.3 Cloud Controls Matrix (CCM)

The CSA Cloud Controls Matrix (CCM) addresses supply chain risk management through its Supply Chain Management, Transparency and Accountability (STA) domain. While the CCM predates the current AI model supply chain threat landscape, its controls around third-party risk assessment, change management, and inventory management are directly applicable to AI model consumption. Organizations should interpret CCM controls on software supply chain risk as encompassing model artifacts, treating model weight files as software dependencies subject to the same acquisition scrutiny, version control, and vulnerability management processes applied to software packages.

## 7.4 Zero Trust Application to Model Loading

CSA's Zero Trust guidance recommends that no resource access be implicitly trusted based on network location or prior authentication. Applied to model loading workflows, zero trust principles support never implicitly trusting a model file because it came from a known registry. Trust should be established through content verification (hash matching), format validation (Safetensors or scanned Pickle), and authorization verification (the model has been approved for use in this pipeline). This verification chain should be enforced at the pipeline level, not left to individual developer discretion, in the same way that zero trust network access is enforced by the access control infrastructure rather than by individual users' judgment about whether to use a VPN.

# 8. Conclusions and Recommendations

The evidence reviewed in this paper supports a clear conclusion: the AI model repository has become an attack surface of strategic interest to threat actors, and the current state of defensive investment does not match the scale of the risk. The attacks are not theoretical—they have been documented at scale, with hundreds of thousands of affected downloads, cloud infrastructure compromise, and a growing catalog of techniques ranging from Pickle exploits to namespace hijacking to agentic skill poisoning. The structural properties of the dependency graph—name-addressed resolution, high fan-out from concentrated namespaces, binary opacity, and the absence of enforced content verification—create the conditions under which individual compromises propagate into systemic events.

The path forward requires action across multiple categories of stakeholder. Model registry operators have begun investing in scanning infrastructure and format migration advocacy; these efforts should be expanded and made universal rather than optional. Framework maintainers should treat model loading security as a first-class design criterion, as the PyTorch team began doing with Safetensors promotion and the `weights_only` parameter—though CVE-2025-32434 demonstrates that this work is not complete. AI-consuming organizations must extend their software supply chain programs to cover model artifacts with the same discipline applied to software packages. And the emerging agentic skill ecosystem must not repeat the early-stage mistakes of npm and PyPI; registry operators should implement baseline security controls before the ecosystem reaches a scale at which remediation becomes structurally difficult.

The following recommendations represent a prioritized defensive action plan, organized by time horizon.

**Immediate Actions (Within 30 Days)**

Organizations should audit all production AI pipelines and identify every model that is loaded from an external registry by floating name rather than a pinned content hash. Each such reference is an exposure to namespace hijacking. Migrate those references to hash-pinned identifiers using the registry's commit hash or digest identifier. Organizations running PyTorch versions prior to 2.6.0 should update immediately to address CVE-2025-32434. Model serving infrastructure—TorchServe, Triton, Ray Serve—should be audited for exposure of management interfaces to untrusted networks and patched to current versions.

**Short-Term Mitigations (30–90 Days)**

Integrate model scanning into all model acquisition pipelines as a mandatory gating control. ModelScan or an equivalent tool should execute for every model file downloaded from an external registry; pipelines should reject or quarantine models that fail scanning. Establish a policy requiring Safetensors format for new model

adoptions; create exceptions processes for Pickle-format models that require isolation and justification. Begin constructing an AI-BOM covering all production model deployments: model name, registry source, content hash, deployment locations, and consuming pipelines.

**Strategic Considerations (90 Days and Beyond)**

Organizations building or expanding agentic AI deployments should establish private skill registries with enforced code signing and organizational approval workflows for skill adoption. Behavioral evaluation pipelines should be developed for models used in security-sensitive contexts, testing model behavior against a catalog of known backdoor trigger patterns and unexpected output distributions. Vendor risk assessments should be extended to cover model providers and AI registry operators as third-party dependencies, with contractual requirements for artifact signing, vulnerability disclosure timelines, and supply chain incident notification. CSA's AICM should be incorporated into AI procurement and governance frameworks as the control reference for model security assessments.

The AI dependency graph is a critical infrastructure component that most organizations have not yet inventoried, secured, or governed with the rigor its role in their systems demands. The incidents of the past two years document what happens when that gap persists: broad, scalable compromise that bypasses perimeter defenses entirely by exploiting the trust relationship between AI systems and the registries they depend on. Closing that gap is the work of the next phase of AI security maturity.

# References


[1] Protect AI and Hugging Face. "4M Models Scanned: Protect AI + Hugging Face 6 Months In." Hugging Face Blog, April 2025.

[2] Koi Security. "OpenClaw ClawHub Malicious Skills Supply Chain Attack." PointGuard AI Security Incidents (citing Koi Security audit), 2026.

[3] Snyk. "ToxicSkills: Snyk Finds Prompt Injection in 36%, 1467 Malicious Payloads in a Study of Agent Skills Supply Chain Compromise." Snyk Blog, 2026.

[4] Palo Alto Networks Unit 42. "Model Namespace Reuse: An AI Supply-Chain Attack Exploiting Model Name Trust." Unit 42 Research, 2025.

[5] CSO Online. "Malicious Hugging Face Model Masquerading as OpenAI Release Hits 244K Downloads." CSO Online, 2025.

[6] The Hacker News. "New Hugging Face Vulnerability Exposes AI Models to Supply Chain Attacks." The Hacker News, February 2024.

[7] Azer Koçulu. "I've Just Liberated My Modules." Kod Fabrik Journal, March 2016. (Referenced as historical parallel; contemporary relevance established by subsequent npm supply chain incident documentation.)

[8] Trail of Bits. "Sleepy Pickle: AI Model Compromise Through Malicious Pickle Files." Trail of Bits Blog, June 2024.

[9] Rapid7. "From .pth to p0wned: Abuse of Pickle Files in AI Model Supply Chains." Rapid7 Blog, 2024.

[10] JFrog Security Research. "Data Scientists Targeted by Malicious Hugging Face ML Models with Silent Backdoor." JFrog Blog, 2024.

[11] NIST National Vulnerability Database. "CVE-2025-32434 Detail." NVD, April 2025.

[12] Kaspersky. "Critical Vulnerability in PyTorch Framework." Kaspersky Blog, 2025.

[13] Sonatype. "Dependency Hijacking Attack Breaches 35 Companies in Exploit." Sonatype Blog, 2021.

[14] Trend Micro. "Malicious OpenClaw Skills Used to Distribute Atomic MacOS Stealer." Trend Micro Research, 2026. *Note: URL requires human verification; 403 response may indicate bot protection.*

[15] Cybersecurity News. "Hackers Leveraged Hugging Face and ClawHub With 575+ Malicious Skills to Deploy Malware." Cybersecurity News, 2026.

[16] Oligo Security. "ShellTorch Explained: Multiple Vulnerabilities in PyTorch Model Server." Oligo Security Blog, October 2023.

[17] Yiming Li et al. "Backdoor Learning: A Survey." IEEE Transactions on Neural Networks and Learning Systems, 2022. arXiv:2007.08745.

[18] IBM X-Force. "What OpenClaw Reveals About Agentic AI Security Risks." IBM Think Blog, 2026. *Note: URL requires human verification; 403 response may indicate bot protection.*

[19] Hugging Face. "Safetensors Audited as Really Safe and Becoming the Default." Hugging Face Blog, 2023.

[20] Protect AI. "ModelScan: Protection Against Model Serialization Attacks." GitHub, 2024.

[21] CISA and G7 Cyber Expert Group. "Software Bill of Materials for AI – Minimum Elements." CISA, 2025.

[22] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 2025.

[23] Cloud Security Alliance. "AI Controls Matrix (AICM)." CSA, 2025.