Secure Infrastructure for Deep Learning Model Deployment: A Survey of Access Control, Runtime Security, and Operational Best Practices

LOVINA DMELLO, NVIDIA Corporation, USA

As organizations increasingly deploy deep learning models in real-world applications, security challenges emerge that extend far beyond traditional software protection. Adversaries can manipulate inputs to change predictions [5, 7], extract models through API abuse, poison training data, or breach privacy [8]. Complex deployment environments involving multiple interconnected services in cloud infrastructure amplify these risks [3].

This survey examines security practices for production deep learning deployments by synthesizing 68 papers across machine learning, systems security, and operations research. A systematic threat taxonomy maps seven categories of attacks to their defensive mechanisms, enabling practitioners to assess security coverage comprehensively. The analysis is organized around three pillars: **Access Control** (authentication, authorization, multi-tenant isolation) [1, 2], **Runtime Security** (threat detection, input validation, monitoring) [4, 27], and **Operational Best Practices** (MLSecOps, regulatory compliance, incident response) [5, 6, 9].

The findings reveal critical gaps between research and practice. Many security mechanisms introduce prohibitive performance overhead (15-30% for adversarial detection), configuration complexity leads to security failures despite available features, and tooling for ML-specific security remains immature [10, 12]. For practitioners, this survey offers actionable guidance through mechanism comparison tables, deployment decision frameworks, and evidence-based recommendations. For researchers, it identifies high-priority directions: efficient defenses with ¡5% overhead, automated security configuration, and production-ready anomaly detection systems. By bridging fragmented literature on ML security, this work provides the first infrastructure-focused view of production ML deployment security.

CCS Concepts: • **Security and privacy** → **Systems security**; **Security services**; • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → *Cloud computing*; • **Software and its engineering** → *Software development process management*.

Additional Key Words and Phrases: Machine learning security, deep learning infrastructure, MLOps, MLSecOps, access control, runtime security, container security, Kubernetes security, model deployment, adversarial machine learning, privacy-preserving machine learning, regulatory compliance, GDPR, multi-tenant isolation

ACM Reference Format:
Lovina Dmello. 2026. Secure Infrastructure for Deep Learning Model Deployment: A Survey of Access Control, Runtime Security, and Operational Best Practices. *ACM Comput. Surv.* 1, 1, Article 1 (February 2026), 29 pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX

Author's Contact Information: Lovina Dmello, ldmello@nvidia.com, lovina.dmello.06@gmail.com, NVIDIA Corporation, Santa Clara, California, USA.

ACM 1557-7341/2026/2-ART1
https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 Introduction

1.1 Background and Motivation

Consider a healthcare organization deploying a deep learning model for cancer detection in medical images. After achieving 95% accuracy in testing, the model receives enthusiastic approval for production deployment. Within weeks, however, the security team discovers something troubling: unauthorized access to patient imaging data through a misconfigured API endpoint. Worse, the incident response team finds evidence that attackers systematically queried the model to extract its parameters—effectively stealing months of expensive research and development [2]. The investigation reveals another problem: the containerized inference service shared resources with other applications in ways that violated HIPAA compliance requirements [3, 9]. While hypothetical, this scenario combines real security failures that have occurred across healthcare, financial, and technology organizations deploying deep learning models [7].

1.2 The Problem

These failures stem from a fundamental mismatch: organizations apply traditional security practices to systems with fundamentally different characteristics [6]. Conventional applications run deterministic code with well-defined security boundaries. In contrast, deep learning systems process inputs through learned patterns where distinguishing normal from malicious behavior becomes probabilistic rather than binary [5].

The differences extend to what needs protection. Traditional software guards well-defined assets like files, databases, and API endpoints. ML systems must also protect model weights, training data, and prediction patterns—assets that lack clear boundaries in conventional security frameworks [2, 8]. Infrastructure requirements differ too. While conventional deployments run in relatively static environments, ML workloads scale dynamically, share hardware resources across tenants, and evolve continuously as models are retrained [11, 12].

1.3 Why This Problem is Important

Security challenges permeate every layer of the deployment stack. At the network boundary, API gateways face a dual challenge: authenticating legitimate requests while detecting queries designed to steal model information [1]. Load balancers must distribute traffic efficiently yet prevent attackers from using prediction patterns to infer training data [8]. The application layer presents its own tensions—inference servers need to validate inputs for adversarial manipulation without introducing latency that degrades user experience [10, 12].

Container orchestrators must balance competing demands: isolating workloads from different tenants while enabling the resource sharing that makes ML economically viable [3, 4]. Storage systems face similar tradeoffs, protecting model weights and datasets without throttling the I/O performance that training and inference demand. Even monitoring becomes more complex, requiring systems to track ML-specific metrics that traditional approaches ignore—prediction confidence distributions, inference latency patterns, per-model resource utilization [13].

Technical complexity aside, organizational challenges compound these difficulties. Security teams bring expertise in network defense, access control, and compliance, but often lack deep learning knowledge [7]. Data science teams understand models intimately yet rarely receive security training. DevOps engineers can deploy containers and manage Kubernetes clusters but may miss ML-specific vulnerabilities [3]. The problem deepens because each group operates with different priorities: security teams emphasize protection, data scientists focus on accuracy, and operations values reliability above all. Effective ML security demands all three perspectives working in concert, yet most organizations lack frameworks to enable this collaboration [5, 6].

Research communities have made progress on individual pieces of this puzzle. Computer security researchers developed techniques for detecting adversarial examples [5]. ML researchers proposed privacy-preserving training methods [8]. Systems researchers improved container isolation mechanisms [3]. However, these advances remain scattered across different communities and venues. What organizations need—and currently lack—is comprehensive guidance that weaves together access control, runtime protection, and operational practices into a cohesive security framework for production ML systems [6].

1.4 Survey Scope and Approach

This survey aims to provide that comprehensive view. The analysis is organized around three pillars that span the full ML deployment lifecycle [5, 6]:

Access Control examines who can use the system, covering API authentication, permission management, and multi-tenant isolation [1, 2]. This pillar explores how traditional access control concepts must adapt to ML-specific operations and assets—protecting not just data and APIs, but also model weights and prediction patterns.

Runtime Security focuses on protecting systems during operation, addressing malicious input detection, behavioral monitoring, and data validation [4, 13]. The emphasis here is on techniques that maintain security without sacrificing the performance characteristics that production ML systems demand [10, 12].

Operational Best Practices considers day-to-day security management, including MLSecOps practices, regulatory compliance, and incident response [5, 7, 9]. This pillar addresses a critical question: how can organizations bridge the expertise gap between security, data science, and operations teams?

1.5 Contributions

This work makes four key contributions:

(1) **Comprehensive Infrastructure-Centric Analysis:** This is the first survey to focus specifically on production infrastructure security for ML deployments, synthesizing 68 papers across machine learning, systems security, and operations research. The analysis covers containers, orchestration platforms, serving frameworks, and deployment pipelines—the infrastructure layers where security failures most commonly occur in practice.

(2) **Integrated Defense-in-Depth Framework:** Rather than examining security mechanisms in isolation, this survey demonstrates how access control, runtime security, infrastructure protection, and operational practices must integrate for effective ML security. A comprehensive taxonomy organizes 45+ security mechanisms while identifying critical dependencies between layers—for instance, how infrastructure isolation enables access control, or how runtime monitoring can inform authorization decisions.

(3) **Quantified Gap Analysis:** The survey identifies and quantifies five critical gaps between research and practice: performance overhead that makes certain defenses impractical (15-30% for adversarial detection), configuration complexity that leads to security failures despite available features, immature ML-specific tooling, organizational expertise gaps between security and ML teams, and emerging threats from large language models and edge deployments.

(4) **Actionable Practitioner Guidance:** Beyond identifying problems, this work provides evidence-based decision support through comparative analysis tables (authentication mechanisms, authorization models, performance overhead), quantified performance data across 15+ security mechanisms, and prioritized research directions organized by technical scope and potential impact.

Practitioners will find actionable patterns for securing ML deployments along with clear guidance on mechanism selection and tradeoffs. Researchers will discover high-priority directions where advances can have immediate practical impact.

1.6 Paper Organization

The remainder of this survey is organized as follows: Section 2 provides background on deep learning deployment infrastructure and surveys related work. Sections 3 through 6 examine our three security pillars: Section 3 covers access control mechanisms (authentication, authorization, multi-tenant isolation, IAM, and API security), Section 4 addresses runtime security (input validation, anomaly detection, threat detection, logging, and rate limiting), Section 5 discusses infrastructure security (container and orchestration security, network security, data protection, and deployment pipelines), and Section 6 explores operational best practices (MLSecOps, regulatory compliance, incident response, and security culture). Section 7 presents our comprehensive taxonomy of security mechanisms. Section 8 discusses critical challenges including the configuration-features tradeoff, performance-security balance, and research-practice gaps. Section 9 concludes with key takeaways for practitioners, prioritized research directions, and closing remarks on the future of ML deployment security.

1.7 Key Acronyms

This survey uses numerous acronyms common in ML deployment security. Table 1 lists the most frequently used terms for quick reference.

1.8 Survey Methodology

This survey synthesizes research on security practices for production ML deployment through systematic literature review. We describe our methodology to enable reproducibility and clarify scope.

1.8.1 Literature Search Strategy. We conducted searches across multiple databases and sources:

Academic Databases: ACM Digital Library, IEEE Xplore, arXiv, and Google Scholar formed our primary sources. These cover the intersection of machine learning, systems security, and software engineering communities relevant to ML deployment security.

Search Terms: We used combinations of terms capturing our focus areas:

- ML security: "machine learning security", "deep learning security", "ML deployment security", "model security"
- Infrastructure: "container security", "Kubernetes security", "ML infrastructure", "model serving security"
- Operations: "MLOps security", "MLSecOps", "ML operations", "secure ML deployment"
- Specific mechanisms: "adversarial ML production", "model extraction", "ML access control", "ML privacy deployment"

Time Period: We focused on papers from 2015-2024, chosen because: (1) containerization became mainstream for ML deployment around 2015, (2) production ML security research accelerated from 2015 onwards, and (3) earlier work focused primarily on training security rather than deployment. We included seminal earlier papers (e.g., 2014 adversarial ML work) when foundational to deployment security.

1.8.2 Inclusion and Exclusion Criteria. **Inclusion Criteria:** Papers were included if they addressed:

- Production deployment of ML models (not just training security)
- Infrastructure-level security (containers, orchestration, serving, networking)
- Access control, runtime security, or operational practices for ML systems
- Practical deployment considerations (performance, scalability, operational complexity)
- Peer-reviewed publications or high-quality technical reports from reputable venues

Exclusion Criteria: We excluded papers that:

- Focused solely on model-level attacks without deployment context
- Addressed only training-time security (data poisoning, backdoors) without deployment implications
- Were purely theoretical without consideration for production constraints
- Lacked sufficient technical depth or evaluation
- Were duplicate publications (we included the most comprehensive version)

1.8.3 Selection Process. Our selection proceeded in three stages:

Stage 1 - Initial Search: Database searches yielded approximately 180 papers matching our search terms.

Stage 2 - Title and Abstract Screening: We screened titles and abstracts against inclusion/exclusion criteria, reducing to 95 papers with clear relevance to ML deployment security.

Stage 3 - Full Paper Review: We conducted full-text review of the 95 papers, selecting 68 papers with substantial contributions to ML deployment infrastructure security. Papers were excluded at this stage if they lacked deployment focus, duplicated coverage from stronger papers, or provided insufficient technical depth.

1.8.4 Synthesis Approach. Rather than cataloging papers individually, we synthesize findings thematically around our three-pillar framework (Access Control, Runtime Security, Operational Practices). For each mechanism or practice, we integrate insights from multiple papers, identify consensus and disagreements, note gaps between research and practice,

Table 1. Key Acronyms Used in This Survey

Acronym	Definition	Acronym	Definition
ABAC	Attribute-Based Access Control	ML	Machine Learning
API	Application Programming Interface	MLOps	Machine Learning Operations
CI/CD	Continuous Integration/Deployment	MLSecOps	Secure Machine Learning Operations
DAC	Discretionary Access Control	mTLS	Mutual Transport Layer Security
DPIA	Data Protection Impact Assessment	RBAC	Role-Based Access Control
GDPR	General Data Protection Regulation	SDN	Software-Defined Networking
HIPAA	Health Insurance Portability & Accountability Act	SSO	Single Sign-On
IAM	Identity and Access Management	TLS	Transport Layer Security
JWT	JSON Web Token	VLAN	Virtual Local Area Network
MAC	Mandatory Access Control	VM	Virtual Machine
MFA	Multi-Factor Authentication	VPC	Virtual Private Cloud

and provide comparative analysis where multiple approaches exist.

1.8.5 Limitations and Potential Biases. We acknowledge several limitations:

Language Bias: We included only English-language publications, potentially missing relevant work in other languages.

Publication Bias: We focused on peer-reviewed academic venues and high-quality technical reports. Gray literature (blog posts, internal reports, vendor documentation) was consulted for context but not systematically included, potentially underrepresenting production practices.

Recency Bias: Rapid evolution in ML deployment means recent papers are overrepresented. We balanced this by including foundational work where relevant.

Infrastructure Focus: Our deliberate focus on deployment infrastructure means we provide less depth on model-level security (adversarial examples, model extraction) already well-covered in existing surveys. This is intentional but represents a scope limitation.

Cloud-Centric: Much research focuses on cloud deployment; edge and on-premises deployments receive less attention in the literature and consequently in our survey.

Despite these limitations, we believe our systematic approach provides comprehensive coverage of production ML deployment security as reflected in current research and practice.

2 Background and Related Work

This section provides essential background on deep learning deployment infrastructure and surveys related work across three key dimensions: access control mechanisms, runtime security approaches, and operational best practices for production ML systems.

2.1 Deep Learning Deployment Infrastructure

2.1.1 Containerization and Orchestration. Container-based virtualization has emerged as the dominant paradigm for deploying deep learning models in production environments [3]. Unlike traditional virtual machines, containers share the host operating system kernel while providing isolated execution environments for applications. This lightweight approach enables faster startup times (50 milliseconds versus 30-40 seconds for VMs) and better resource utilization, making containers ideal for deploying microservice-based ML inference systems [3].

Kubernetes has become the de facto standard for container orchestration, providing automated deployment, scaling, and management of containerized applications [4]. In ML serving contexts, Kubernetes manages multiple model instances across clusters, handles load balancing, and provides self-healing capabilities through automated restarts and health checks. However, this orchestration complexity introduces security challenges. Containers share the host kernel, creating potential attack vectors where malicious processes in one container could exploit kernel vulnerabilities to compromise the host or other containers [3]. The multi-tenant nature of Kubernetes clusters, where multiple teams or applications share infrastructure, amplifies these risks.

2.1.2 Model Serving Frameworks. Modern ML serving frameworks provide the infrastructure layer between trained models and production applications. TensorFlow Serving, introduced by Google, pioneered flexible, high-performance model serving with support for model versioning, A/B testing, and dynamic model loading [10]. These frameworks must balance competing requirements: low-latency inference for real-time applications, high throughput for batch processing, resource efficiency for cost optimization, and security isolation between different models and tenants [11].

Recent work on power-aware model serving demonstrates the performance-security tradeoff inherent in ML deployment [12]. Security mechanisms such as input validation, request authentication, and monitoring introduce computational overhead that impacts inference latency and energy consumption. This creates tension between security requirements and the performance expectations of production ML systems.

2.2 Access Control for ML Systems

2.2.1 Traditional Access Control Models. Access control in computing systems has evolved through several paradigms. Discretionary Access Control (DAC) allows resource owners to grant permissions, while Mandatory Access Control (MAC) enforces system-wide policies based on security classifications. Role-Based Access Control (RBAC) assigns permissions to roles rather than individual users, simplifying administration in large organizations. Attribute-Based Access Control (ABAC) makes decisions based on attributes of users, resources, and environmental context, providing more fine-grained control.

However, these traditional models were designed for conventional software systems with well-defined access points and static resources. ML systems present fundamentally different characteristics: model weights are sensitive assets requiring protection, prediction APIs can leak information about training data, and the probabilistic nature of ML predictions creates new attack surfaces [2].

2.2.2 ML-Specific Access Control Challenges. Applying machine learning to enhance access control systems has been an active research area. ML techniques can automate policy mining from access logs, detect anomalous access patterns, and adapt policies dynamically based on context [1, 2]. Supervised learning approaches classify access requests as legitimate or suspicious, while reinforcement learning enables adaptive policies that learn from access patterns over time [1].

However, securing access to ML systems themselves requires different approaches. Research has identified several ML-specific access control requirements [2]: protecting model intellectual property through controlled API access, preventing model extraction attacks where adversaries reconstruct models through repeated queries, isolating training data from inference systems to prevent data leakage, and managing multi-tenant scenarios where multiple organizations share ML infrastructure while maintaining data privacy.

2.3 Runtime Security and Threat Detection

2.3.1 Adversarial Attacks on ML Systems. Deep learning models are vulnerable to adversarial attacks where carefully crafted inputs cause misclassification. These attacks exploit the high-dimensional input space and the model's learned decision boundaries. Adversarial examples can be generated through gradient-based methods that iteratively modify inputs to maximize prediction error while remaining imperceptible to humans. In production deployments, such attacks could have serious consequences: autonomous vehicles misclassifying stop signs, medical diagnosis systems providing incorrect predictions, or financial fraud detection systems failing to identify fraudulent transactions [5].

Beyond adversarial examples, ML systems face model extraction attacks (stealing model weights through API queries), data poisoning (corrupting training data to backdoor models), and membership inference attacks (determining if specific data was used in training) [8]. These attacks are amplified in cloud deployments where multiple tenants share infrastructure and adversaries may have increased access to system internals.

2.3.2 Container Security and Monitoring. The containerized nature of modern ML deployments introduces specific security concerns. Research on container security has identified four key threat scenarios [3]: protecting containers from malicious applications inside them, inter-container protection to prevent lateral movement, protecting the host system from compromised containers, and protecting containers from malicious or semi-honest host operators.

Linux kernel features such as namespaces, control groups (cgroups), capabilities, and seccomp provide the foundation for container isolation. However, vulnerabilities in these mechanisms or misconfigurations can compromise security. Recent work on detecting cryptomining malware in Kubernetes clusters demonstrates the importance of monitoring system calls and resource usage patterns to identify anomalous behavior [4]. Such monitoring systems must balance security detection capabilities with the performance overhead of continuous observation [13].

2.4 Operational Security and MLOps

2.4.1 MLOps and Security Integration. Machine Learning Operations (MLOps) emerged as a practice to streamline the ML lifecycle from development through deployment and maintenance. However, early MLOps practices focused primarily on automation and reliability, with security often added as an afterthought [6]. This gap has led to security incidents where ML models were deployed with inadequate access controls, insufficient input validation, or missing monitoring capabilities.

The concept of Secure MLOps (MLSecOps) addresses this by embedding security throughout the ML lifecycle [5]. This includes secure data collection and storage, privacy-preserving training techniques, secure model serialization and storage, authenticated and authorized model serving, continuous monitoring for adversarial attacks and data drift, and incident response procedures for ML-specific threats. Research has identified key challenges in implementing MLSecOps [6]: organizational gaps between security, data science, and operations teams; lack of standardized security practices for ML systems; difficulty balancing security controls with performance requirements; and limited tools for ML-specific security monitoring.

2.4.2 Compliance and Privacy. Deploying ML systems with personal or sensitive data requires compliance with regulations such as GDPR (General Data Protection Regulation)

and HIPAA (Health Insurance Portability and Accountability Act). These regulations impose requirements for data minimization, purpose limitation, transparency, and accountability [9]. ML systems create additional compliance challenges because models can memorize and leak training data through their predictions.

Privacy-preserving machine learning techniques such as differential privacy, federated learning, and secure multi-party computation offer mathematical guarantees about information leakage. However, applying these techniques in production involves tradeoffs between privacy guarantees, model accuracy, and computational cost [8]. Organizations must perform Data Protection Impact Assessments (DPIAs) to quantify privacy risks from ML models and implement appropriate safeguards [8].

Recent research has developed tools to measure privacy risks in deployed ML models, enabling organizations to assess whether their systems meet regulatory requirements [8]. However, the gap between research prototypes and production-ready privacy-preserving ML systems remains significant. Organizations need practical guidance on implementing privacy controls that satisfy regulatory requirements while maintaining acceptable model performance [9].

2.4.3 Emerging Security Strategies. Recent work on MLOps-enabled security strategies for operational technologies demonstrates the evolution toward more comprehensive security frameworks [7]. These frameworks recognize that ML security requires integration across multiple layers: infrastructure security (container and network isolation), application security (API authentication and input validation), model security (adversarial robustness and privacy preservation), and operational security (monitoring, logging, and incident response).

However, significant gaps remain in bridging research advances with production practices. Many proposed security techniques lack evaluation in real-world deployment contexts, fail to account for performance constraints, or require expertise that organizations lack [6]. This survey aims to address these gaps by providing a comprehensive view of security practices across the full ML deployment lifecycle, identifying critical tradeoffs, and offering practitioners actionable guidance for building secure ML systems.

2.5 Related Surveys and Positioning

Several surveys have examined aspects of ML security and deployment, but none provide the comprehensive infrastructure-focused view we present here.

2.5.1 ML Security Surveys. Previous surveys on adversarial machine learning have focused primarily on attack and defense techniques at the model level, examining adversarial example generation, model extraction, and data poisoning. While these works provide valuable taxonomies of ML-specific attacks, they typically treat deployment infrastructure as an implementation detail rather than a central security concern.

Surveys on privacy-preserving machine learning have examined techniques like differential privacy, federated learning, and secure computation [8]. However, these focus on privacy during training and inference, with limited discussion of how privacy mechanisms integrate into production deployment pipelines or how they interact with infrastructure security controls.

2.5.2 Systems Security Surveys. Research on container security has provided comprehensive analyses of isolation mechanisms, vulnerabilities, and attack vectors in containerized environments [3]. However, these surveys target general-purpose containerized applications and do not address ML-specific security requirements such as protecting model weights, detecting adversarial inputs, or preventing model extraction through API abuse.

Surveys on cloud security and multi-tenant systems examine isolation, access control, and monitoring in cloud environments. While relevant to ML deployment, these works do not address the unique characteristics of ML workloads: dynamic scaling based on inference load, resource sharing between training and serving, or the probabilistic nature of ML predictions that complicates anomaly detection.

2.5.3 MLOps Surveys. Recent surveys on MLOps practices have documented the challenges organizations face in operationalizing ML systems [6]. These works identify gaps in tooling, organizational practices, and standardization. However, security typically appears as one concern among many, without deep analysis of the specific security challenges in ML deployment or comprehensive guidance on security controls.

Work on MLSecOps has begun integrating security into the ML lifecycle [5], proposing frameworks that embed security from development through deployment. However, existing MLSecOps research focuses primarily on secure development practices and model-level security, with less emphasis on production infrastructure security, multi-tenant isolation, and operational security monitoring.

2.5.4 Positioning of This Survey. Table 2 positions our work relative to existing surveys across five key dimensions. While prior surveys provide deep analysis of specific aspects—adversarial ML, container security, privacy, access control, or MLOps—none offer comprehensive infrastructure-focused coverage of production ML deployment security.

Our survey differs from prior work in several key aspects:

Infrastructure-Centric View: We center our analysis on the deployment infrastructure—containers, orchestration, serving frameworks, and monitoring systems—rather than treating infrastructure as secondary to model-level security. This reflects the reality that production ML security failures

Table 2. Comparison of Related Surveys on ML Security and Deployment

Survey Focus	Model-Level Security	Infrastructure Security	Access Control	Operational Practices	Production Focus
Adversarial ML Surveys [27]	Comprehensive	Minimal	Limited	Minimal	Research prototypes
Container Security [3]	Not addressed	Comprehensive	Limited	Minimal	General apps
ML Privacy Surveys [8]	Privacy-focused	Minimal	Limited	Limited	Training-focused
Access Control [2]	Minimal	Not addressed	Comprehensive	Minimal	Conceptual
MLOps Surveys [6]	Limited	Limited	Limited	Comprehensive	Operations-focused
SecMLOps Framework [5]	Moderate	Moderate	Moderate	Comprehensive	Framework proposal
This Survey	**Moderate**	**Comprehensive**	**Comprehensive**	**Comprehensive**	**Infrastructure-focused**

often stem from infrastructure misconfigurations, inadequate access controls, or insufficient monitoring rather than sophisticated adversarial attacks on models.

End-to-End Deployment Lifecycle: Rather than focusing on a single aspect (attacks, defenses, privacy, or operations), we examine security across the complete deployment lifecycle. We show how access control, runtime security, and operational practices must work together to provide defense-in-depth.

Practitioner-Focused Guidance: We bridge the gap between academic research and production deployment by examining not just what security mechanisms exist, but how they can be implemented in real systems given performance constraints, organizational limitations, and tool maturity. We highlight practical tradeoffs rather than idealized security models.

Comprehensive Citation of Practice: We systematically review not only academic literature but also industry practices, tool documentation, and operational experience reports. This provides a more complete picture of the state of ML deployment security as practiced in production environments.

Integration of Multiple Communities: ML security research spans multiple communities—machine learning, systems security, software engineering, and operations—that often work in isolation. We integrate insights across these communities to provide a unified view of ML deployment security.

This comprehensive approach enables us to identify critical gaps where stronger security mechanisms are needed, highlight areas where existing techniques remain immature for production use, and provide actionable guidance for practitioners building secure ML systems.

2.6 Threat Landscape for ML Deployments

Before examining security mechanisms, we systematically characterize the threat landscape for ML deployment infrastructure. Understanding what threats exist, their attack vectors, and potential impacts provides the foundation for evaluating defensive mechanisms examined in subsequent sections.

2.6.1 Threat Taxonomy. ML deployment infrastructure faces threats spanning multiple dimensions. We organize threats into seven primary categories based on attack objectives and target assets.

1. Adversarial Input Attacks target ML models through carefully crafted inputs designed to cause misclassification while appearing legitimate. Attackers manipulate pixel values in images, perturb features in structured data, or craft text inputs to exploit model vulnerabilities [5, 27]. These attacks succeed because models learn decision boundaries that can be exploited through imperceptible perturbations. Attack vectors include direct API access where attackers query models with crafted inputs, or indirect attacks where adversarial examples are introduced into normal data pipelines. Impact ranges from individual misclassifications (security camera misidentifying objects) to systematic failures (spam filters disabled by crafted messages). Detection difficulty is high: adversarial examples often satisfy all input validation constraints while exploiting model-specific vulnerabilities. Addressed in Section 4 (Runtime Security).

2. Model Extraction and Theft involves attackers reconstructing model parameters or functionality through API queries or infrastructure access. Model extraction attacks query prediction APIs systematically to train surrogate models mimicking original behavior [2]. Model theft steals model weights directly through compromised infrastructure, misconfigured storage, or insider access. Attack vectors include API abuse (high-volume queries to infer model behavior), storage compromise (accessing model weight files), and memory extraction (reading GPU memory during inference). Impact includes intellectual property loss (months of research stolen),

competitive disadvantage, and enabling subsequent attacks (adversarial examples crafted for stolen models). Detection difficulty is moderate for API-based extraction (detectable through query pattern analysis) but high for direct theft (appears as legitimate access). Addressed in Sections 3 (Access Control) and 4 (Runtime Security).

3. Data Poisoning and Supply Chain Attacks corrupt training data or introduce backdoors during model development. Data poisoning injects malicious examples into training data, causing models to learn attacker-desired behaviors [27]. Supply chain attacks compromise model registries, pretrained models, or training pipelines. Attack vectors include training data compromise (injecting poisoned examples), dependency compromise (malicious packages in training code), and model registry attacks (replacing legitimate models with backdoored versions). Impact includes backdoored models that misbehave on attacker-chosen triggers while performing normally otherwise, and complete model compromise requiring retraining. Detection difficulty is very high: poisoned data often appears legitimate, backdoors activate only on specific triggers, and supply chain compromises may persist across deployments. Addressed in Sections 5 (Infrastructure Security) and 6 (Operational Best Practices).

4. Privacy Breaches and Information Leakage extract sensitive information about training data through model queries or access. Membership inference attacks determine whether specific examples were used in training [8]. Model inversion reconstructs training data features from model outputs. Training data exposure occurs through misconfigured storage or insufficient access controls. Attack vectors include statistical analysis of model predictions (inferring training data properties), API queries designed to leak information, and direct storage access (reading training datasets). Impact includes regulatory violations (GDPR, HIPAA breaches), competitive intelligence loss, and exposure of sensitive personal or proprietary information. Detection difficulty is moderate to high: statistical attacks appear as normal API usage, requiring sophisticated analysis to detect. Addressed in Sections 3 (Access Control), 4 (Runtime Security), and 6 (Operational Best Practices, Compliance).

5. Infrastructure Compromise and Lateral Movement exploits vulnerabilities in deployment infrastructure to access models, data, or other systems. Container escape attacks break out of container isolation to compromise hosts [3]. Privilege escalation exploits misconfigurations to gain elevated access. Lateral movement spreads from initial compromise to additional systems. Attack vectors include container vulnerabilities (kernel exploits enabling escape), misconfigured network policies (allowing unauthorized communication), credential compromise (stolen API keys or certificates), and software vulnerabilities (unpatched systems). Impact ranges from single service compromise to full infrastructure takeover, data exfiltration, and persistent access. Detection difficulty varies: sophisticated attacks evade detection while noisy attacks trigger monitoring. Addressed in Section 5 (Infrastructure Security).

6. Denial of Service and Resource Exhaustion prevents legitimate use by overwhelming systems or exhausting resources. Inference overload floods prediction APIs with requests. Resource exhaustion depletes GPU memory, CPU, or network bandwidth. Cost inflation attacks trigger expensive operations (large model inference, GPU allocation) [12]. Attack vectors include API flooding (high-volume requests), expensive query crafting (inputs requiring maximum computation), and resource allocation abuse (requesting maximum instances). Impact includes service unavailability, degraded performance for legitimate users, and inflated operational costs. Detection difficulty is moderate: volumetric attacks are easily detected, but sophisticated low-volume attacks mimicking legitimate traffic are harder to identify. Addressed in Sections 3 (API Security) and 4 (Rate Limiting, Runtime Security).

7. Insider Threats and Misuse stem from authorized users abusing access privileges. Malicious insiders intentionally steal models or data. Negligent insiders accidentally expose sensitive assets through misconfigurations. Credential abuse uses legitimate credentials for unauthorized purposes. Attack vectors include privilege abuse (authorized users accessing resources beyond need), social engineering (tricking users into providing access), and compromised credentials (stolen or shared). Impact includes data exfiltration, model theft, compliance violations, and reputational damage. Detection difficulty is very high: insider actions appear legitimate, requiring behavioral analysis and anomaly detection to identify misuse. Addressed in Sections 3 (Access Control, IAM), 4 (Monitoring), and 6 (Operational Best Practices).

2.6.2 Threat-Defense Mapping. Table 3 maps threats to the security mechanisms examined in this survey. This matrix shows which defensive mechanisms address each threat category, enabling practitioners to assess coverage and identify gaps in their security postures. Note that effective defense requires multiple mechanisms (defense-in-depth): no single mechanism provides complete protection against any threat.

2.6.3 Threat Prioritization for Practitioners. Organizations must prioritize defenses based on their specific threat profiles. We provide guidance based on deployment context:

High-Value Models (IP Protection Priority): Focus on model extraction and theft prevention through strong access control (Section 3), API security with rate limiting and query monitoring (Section 3.5), and infrastructure isolation (Section 5.1). Accept minimal performance overhead for strong authentication and monitoring.

Regulated Industries (Privacy/Compliance Priority): Emphasize privacy breach prevention through encryption (Section 5.3), access control with audit trails (Section 3.1-3.4), and compliance frameworks (Section 6.2). Invest in comprehensive logging for regulatory documentation.

Adversarial Environments (Robustness Priority): Prioritize adversarial input defenses through input validation (Section 4.1), anomaly detection (Section 4.2), and prediction

Table 3. Threat-Defense Matrix for ML Deployment Infrastructure

Threat Category	Primary Defenses	Secondary Defenses	Detection Mechanisms	Survey Coverage
Adversarial Inputs	Input validation, Adversarial detection	Schema checking, Anomaly detection	Prediction monitoring, Confidence analysis	Section 4.1, 4.3
Model Extraction	Rate limiting, API authentication	Query pattern analysis, Response obfuscation	API monitoring, Query analysis	Section 3.5, 4.5
Data Poisoning / Supply Chain	Secure CI/CD, Model signing	Training data validation, Provenance tracking	Integrity checks, Model testing	Section 5.4, 6.1
Privacy Breaches	Access control, Data encryption	Differential privacy, Federated learning	Audit logs, Access monitoring	Section 3.1-3.3, 6.2
Infrastructure Compromise	Container isolation, Network segmentation	Vulnerability scanning, Patch management	Intrusion detection, Log analysis	Section 5.1, 5.2
Denial of Service	Rate limiting, Resource quotas	Load balancing, Autoscaling	Traffic monitoring, Resource metrics	Section 3.5, 4.5
Insider Threats	RBAC/ABAC, MFA	Behavior analytics, Audit trails	Anomaly detection, Access logs	Section 3.2, 3.4, 4.2

monitoring (Section 4.4). Balance detection accuracy against performance overhead based on risk tolerance.

Multi-Tenant Platforms (Isolation Priority): Focus on infrastructure compromise prevention through strong container isolation (Section 5.1), network segmentation (Section 5.2), and tenant separation (Section 3.3). Implement defense-in-depth assuming some isolation mechanisms may fail.

Cost-Sensitive Deployments (DoS Prevention Priority): Emphasize denial of service prevention through rate limiting (Section 4.5), resource quotas (Section 3.5), and cost monitoring. Implement per-tenant usage tracking and automated alerts for anomalous consumption.

This threat landscape provides the foundation for understanding security mechanisms examined in subsequent sections. As we discuss each mechanism, we reference which threats it addresses and its effectiveness against different attack vectors.

3 Access Control for Deep Learning Deployments

Access control forms the first line of defense in securing ML deployment infrastructure. This section examines authentication mechanisms, authorization models, multi-tenant isolation, identity management, and API security practices specific to production ML systems. Unlike traditional applications with well-defined access boundaries, ML systems must protect multiple asset types—model weights, training data, prediction APIs, and infrastructure resources—each requiring tailored access control approaches.

3.1 Authentication Mechanisms

Authentication verifies the identity of entities accessing ML systems. Production ML deployments involve multiple authentication scenarios: users accessing prediction APIs, services communicating within the ML pipeline, administrators managing infrastructure, and automated systems triggering model retraining. Each scenario presents unique requirements and constraints.

3.1.1 API-Based Authentication. ML inference services typically expose REST or gRPC APIs for prediction requests. API authentication mechanisms must balance security with performance, as authentication overhead directly impacts inference latency. Common approaches include API keys, JSON Web Tokens (JWT), and OAuth 2.0 flows.

API keys provide simple authentication by embedding secret tokens in requests. However, key management becomes challenging at scale: keys must be rotated regularly, revoked when compromised, and scoped appropriately to limit damage from leakage. Research on API security for ML systems demonstrates that naïve API key implementations often fail to implement proper rate limiting, enabling attackers to extract models through repeated queries. Dynamic machine learning models can assess API access risks in real-time, adapting authentication requirements based on request patterns and user behavior [14].

The work by Nokovic et al. [14] demonstrates that combining qualitative and quantitative verification over models created on training data can significantly reduce false access probability. Their risk assessment framework for API authentication uses digital identity attributes such as IP address, browser user agent, and user activity patterns to estimate risk levels for each authentication attempt. This approach enables adaptive security policies that balance usability with protection against credential stuffing, bot attacks, and account takeover attempts.

OAuth 2.0 and OpenID Connect provide more sophisticated authentication flows suitable for user-facing ML applications. These protocols separate authentication from authorization, enabling integration with enterprise identity providers while maintaining API security. However, implementing OAuth correctly requires careful attention

to token lifetimes, refresh mechanisms, and scope management—complexity that many ML deployment teams lack expertise to handle properly.

3.1.2 Certificate-Based Authentication. For service-to-service communication within ML pipelines, certificate-based authentication using mutual TLS (mTLS) provides stronger security guarantees than shared secrets. Each service receives a cryptographic identity certificate, and connections are authenticated through cryptographic protocols rather than shared passwords. This approach aligns well with microservices architectures common in ML deployments, where models, feature stores, and monitoring services must communicate securely.

Kubernetes provides native support for certificate management through its Certificate API, automating certificate issuance and rotation for workloads. However, certificate-based authentication introduces operational complexity: certificate authorities must be managed, certificates must be rotated before expiration, and revocation mechanisms must handle compromised credentials. Research on container security for ML deployments identifies certificate misconfiguration as a common vulnerability, particularly when teams unfamiliar with PKI infrastructure attempt to implement mTLS.

3.1.3 Multi-Factor and Biometric Authentication. For administrative access to ML infrastructure and sensitive operations like model deployment, multi-factor authentication (MFA) provides additional security layers beyond passwords. MFA combines something the user knows (password), something they have (hardware token or mobile device), and optionally something they are (biometric factors).

Recent work on continuous authentication using behavioral biometrics demonstrates ML's dual role in access control: machine learning techniques can analyze keystroke dynamics, mouse movements, and interaction patterns to continuously verify user identity during sessions [15]. Integration of FastAPI with machine learning for behavioral biometrics achieves near-zero false positive rates while maintaining imperceptible user experience impact. The system performs continuous authentication without interrupting user actions, using neural networks to classify behavior patterns based on click length, button type, and screen interaction sequences.

However, behavioral biometrics raise privacy concerns, requiring careful consideration of data collection policies and compliance with regulations like GDPR [9]. Organizations must balance the security benefits of continuous monitoring against user privacy expectations and regulatory requirements for consent and data minimization.

3.1.4 Authentication Mechanism Comparison. Table 4 summarizes the authentication mechanisms discussed above, comparing their security properties, performance characteristics, implementation complexity, and deployment contexts. This comparison helps practitioners select appropriate authentication approaches based on their specific requirements and constraints.

Different authentication mechanisms suit different contexts. API keys work well for development and internal tools where convenience matters more than strong security guarantees. OAuth/JWT tokens strike a balance between security and usability, making them appropriate for user-facing services. mTLS provides strong cryptographic guarantees ideal for automated service-to-service communication. MFA adds crucial protection for administrative operations that require human oversight, while behavioral biometrics enable continuous authentication in high-security contexts. Most organizations employ multiple mechanisms simultaneously—API keys during development, OAuth for external users, mTLS between internal services, and MFA for administrative access.

3.2 Authorization Models

While authentication verifies identity, authorization determines what authenticated entities can do. ML systems require authorization decisions at multiple levels: which users can invoke which models, which services can access which data sources, which administrators can deploy model updates, and which operations require elevated privileges.

3.2.1 Role-Based Access Control (RBAC). RBAC remains the most widely deployed authorization model in enterprise environments. The concept is straightforward: users belong to roles (e.g., "data scientist," "model operator," "infrastructure admin"), and permissions attach to roles rather than individual users. This approach simplifies administration considerably—instead of managing permissions for thousands of users individually, administrators define a manageable number of roles with appropriate permissions.

For ML deployments, RBAC provides coarse-grained control that aligns well with organizational boundaries. A data science team might receive permissions to train models and submit deployment requests, while an operations team gets permissions to approve deployments and manage infrastructure. RBAC's rigidity becomes problematic, however, when fine-grained control is needed. Trying to define roles that capture ML operations' nuances (e.g., "can deploy models for customer segment A but not segment B") quickly leads to role explosion.

Research on database intrusion detection in RBAC-enabled systems shows that even well-designed RBAC policies remain vulnerable to privilege escalation or role mining attacks [18]. Comparison studies by Soni and Kumar [17] evaluating RBAC and ABAC for private cloud deployments (highly relevant to ML infrastructure) reveal an important tradeoff. RBAC's simplicity comes at the cost of limited expressiveness for complex scenarios. As roles multiply to accommodate diverse access requirements, administrative complexity grows proportionally, eventually negating RBAC's supposed management advantages.

3.2.2 Attribute-Based Access Control (ABAC). ABAC takes a fundamentally different approach, making authorization decisions based on attributes of the user, resource, action,

Table 4. Comparison of Authentication Mechanisms for ML Deployments

Mechanism	Security Level	Performance Impact	Implementation Complexity	Best Use Case
API Keys	Low-Medium	Minimal (¡1%)	Low	Development, internal tools, low-risk APIs
OAuth 2.0/JWT	Medium-High	Low (2-5%)	Medium	User-facing applications, third-party integrations
mTLS Certificates	High	Medium (5-10%)	High	Service-to-service, internal microservices
Multi-Factor (MFA)	High	N/A (human)	Medium	Administrative access, sensitive operations
Behavioral Biometrics	Very High	Low (1-3%)	Very High	Continuous authentication, high-security contexts

and environment context. Instead of static role assignments, ABAC policies express contextual rules—for example, "data scientists can access training data if classification level equals 'internal' and request originates from corporate network." This flexibility makes ABAC particularly well-suited for ML systems with complex, dynamic access requirements.

For ML deployments, ABAC enables policies that adapt intelligently to context. Prediction requests from verified users during business hours might grant access to high-accuracy models, while unverified requests or off-hours access could route to restricted-capacity alternatives. Model deployment policies might require multiple approvals for models trained on sensitive data while allowing automatic deployment for those trained on public datasets. As Soni and Kumar [17] demonstrate, ABAC provides greater flexibility and scalability than RBAC, especially for cloud-based ML infrastructure where access patterns vary dynamically with workload, location, and risk level.

This expressiveness comes with tradeoffs, however. ABAC policies can become quite complex, making them difficult to verify, test, and debug. Machine learning approaches for access control policy verification [16] offer a solution, automatically checking policy logic for inconsistencies, conflicts, and unintended consequences—critically important for ML deployments where incorrect authorization could leak sensitive model information or enable unauthorized data access. The NIST work by Hu [16] proposes using classification algorithms to directly verify the logic of policy rules without requiring comprehensive test cases or system translation, making verification more practical even for complex ABAC policies.

3.2.3 Policy-Based and Dynamic Authorization. Recent work explores dynamic access control policies that adapt based on learned patterns. Reinforcement learning approaches enable policies to optimize security-usability tradeoffs by learning from access patterns: frequently granted requests might be expedited, while unusual patterns trigger additional verification. Blockchain-based approaches provide distributed, tamper-resistant policy enforcement suitable for multi-organization ML collaborations.

Dynamic authorization raises important questions for ML systems: How should policies handle concept drift in user behavior? When should learned policies override explicit rules? How can organizations audit and explain authorization decisions made by ML models? These questions remain active research areas with limited production guidance.

3.2.4 Authorization Model Comparison. Table 5 compares the three primary authorization models for ML deployments. The comparison evaluates policy flexibility, implementation complexity, performance characteristics, and suitability for different deployment contexts. This analysis helps organizations select authorization approaches aligned with their security requirements, operational capabilities, and scalability needs.

Organizations often employ hybrid approaches: RBAC for coarse-grained organizational access, ABAC for fine-grained context-aware policies, and ML-based detection for identifying anomalous access patterns [2, 17]. The choice depends on organizational maturity, compliance requirements, and available security expertise. Teams with limited security resources should start with RBAC and progressively add ABAC policies for specific high-risk operations, deferring ML-based authorization until they gain experience with simpler models.

3.3 Multi-Tenant Isolation

Modern ML platforms frequently serve multiple tenants—different teams, business units, or external customers—sharing underlying infrastructure for cost efficiency. Multi-tenant isolation ensures that tenants cannot access each other's data, models, or resources, despite sharing compute, storage, and network infrastructure.

3.3.1 Namespace and Resource Isolation. Kubernetes provides namespace-based isolation, allowing administrators to partition clusters into logical groups. Each tenant receives a dedicated namespace with resource quotas limiting CPU, memory, and storage consumption. Network policies restrict traffic between namespaces, preventing lateral movement if one tenant's workload is compromised.

However, namespace isolation alone provides insufficient security for ML workloads. Research on multi-tenant security

Table 5. Comparison of Authorization Models for ML Systems

Model	Flexibility	Complexity	Policy Management	Best For
RBAC	Low	Low	Simple, role-based	Small teams, static permissions, clear organizational hierarchy
ABAC	High	High	Complex, attribute rules	Dynamic policies, compliance requirements, context-aware decisions
ML-Based Dynamic	Very High	Very High	Learned policies, adaptive	High-security contexts, pattern-based decisions, anomaly detection

in cloud computing identifies several attack vectors that cross namespace boundaries: container escape vulnerabilities that exploit shared kernel, side-channel attacks that leak information through shared resources, and metadata API abuse that exposes cluster-wide information. Graph-based models for multi-tenant security [19] demonstrate that analyzing resource dependencies reveals isolation violations not apparent from namespace configuration alone.

Pasham [19] shows that graph theory enables structured representation of relationships and interactions between tenants, resources, and services in multi-tenant cloud environments. By modeling cloud resources and tenant interactions as graphs, these security models can effectively monitor risks, detect pre-identified abnormalities, and control cross-tenancy data breaches. Graph-based approaches using community identification and machine learning for anomaly detection prove particularly effective for preventing framework invasions and resource contention in scenarios like VM deployment. However, scalability concerns and integration with traditional security models remain active challenges.

3.3.2 Data Isolation and Encrypted Storage. ML systems process and store sensitive data requiring tenant-specific encryption and access controls. Data isolation mechanisms must protect multiple data types: raw input data, training datasets, model weights, and prediction results. Each type has different sensitivity and access patterns.

Encryption at rest protects stored data from unauthorized access, but key management becomes complex in multi-tenant scenarios: each tenant requires separate encryption keys, keys must be rotated regularly, and access logs must track which tenants accessed which data. Hardware security modules (HSMs) or cloud-native key management services provide centralized key storage, but introduce performance overhead and potential bottlenecks for data-intensive ML workloads.

Recent work on security in multi-tenancy cloud environments specific to ML deployments shows that naive data isolation—simple file system permissions or database access controls—fails under sophisticated attacks. Secure multi-party computation and homomorphic encryption enable computation on encrypted data, allowing tenants to benefit from shared infrastructure without exposing raw data. However, these techniques remain too computationally expensive for most production ML scenarios.

3.3.3 Network Segmentation and Traffic Isolation. Network-level isolation prevents tenants from accessing each other's services or intercepting traffic. Virtual Private Clouds (VPCs), Virtual Local Area Networks (VLANs), or software-defined networking (SDN) create isolated network segments for each tenant. Within Kubernetes, Network Policies enforce firewall-like rules restricting which pods can communicate.

For ML deployments, network segmentation must accommodate complex communication patterns: model serving endpoints must be accessible to tenant users, feature stores must be accessible to inference services, and monitoring systems must collect metrics from all components. Balancing connectivity requirements with isolation constraints requires careful network architecture. Research on ML infrastructure security demonstrates that misconfigured network policies—particularly overly permissive default rules—create vulnerabilities enabling cross-tenant access.

3.4 Identity and Access Management (IAM)

IAM frameworks provide centralized management of identities, authentication, and authorization across ML deployments. Cloud providers offer IAM services (AWS IAM, Azure AD, GCP IAM) that integrate with ML platforms. Enterprise organizations often deploy identity providers like Okta or Active Directory.

3.4.1 Service Accounts and Non-Human Identities. ML deployments involve numerous automated processes: training pipelines that fetch data, model servers that load weights, monitoring services that collect metrics, and CI/CD systems that deploy updates. Each automated process requires credentials to authenticate and authorization to access resources.

Service accounts provide identities for automated processes, with credentials managed separately from user accounts. In Kubernetes, service accounts receive automatically mounted tokens enabling pod-to-API-server authentication. However, service account management introduces challenges: accounts proliferate as ML systems grow, permissions must follow least-privilege principles, and compromised service accounts provide persistent attacker access.

Machine learning in action for securing IAM APIs demonstrates that ML-based risk authentication decision engines can detect anomalous service account behavior [21]. The Risk Authentication/Assessment Decision Engine (RADE) system

by Djosic et al. [21] uses digital identity attributes such as IP address, browser user agent, and user activity to estimate risk levels for each authentication attempt. Their practical implementation shows how ML techniques integrate into DevOps ecosystems to detect unusual access patterns, credential use from unexpected locations, or privilege escalation attempts. These systems adapt to normal behavior patterns, flagging deviations for investigation while minimizing false positives that disrupt legitimate automated workflows.

3.4.2 Secret Management. ML systems require numerous secrets: database passwords, API keys for external services, encryption keys for sensitive data, and credentials for cloud resources. Secrets must be stored securely, accessed only by authorized services, rotated regularly, and audited comprehensively.

Secret management solutions like HashiCorp Vault, Kubernetes Secrets, or cloud-native options (AWS Secrets Manager, Azure Key Vault) provide centralized secret storage with access controls, encryption, and audit logging. However, integrating secret management with ML workflows requires careful design. Secrets must be available when models start, without exposing them in configuration files or environment variables visible to attackers.

Common pitfalls include secrets hardcoded in Docker images, secrets stored in version control, overly broad secret access policies, and insufficient monitoring of secret usage. Research on ML infrastructure security emphasizes that secret management failures enable attack chains. Compromised secrets lead to data breaches, which enable model extraction, which reveals training data leakage.

3.5 API Security for ML Serving

ML prediction APIs form the primary interface between models and applications. API security mechanisms must prevent abuse while maintaining the low latency required for real-time inference.

3.5.1 Rate Limiting and Quota Management. Rate limiting restricts the number of requests users or services can make within time windows, preventing denial-of-service attacks and resource exhaustion. For ML APIs, rate limiting serves additional purposes: preventing model extraction attacks that require many queries, controlling inference costs, and ensuring fair resource sharing among tenants.

Determining appropriate rate limits for ML APIs requires balancing multiple factors. Legitimate batch prediction workloads generate high request volumes, while low-volume but high-value real-time predictions require immediate service. Static rate limits fail to accommodate this diversity. Dynamic limits based on learned usage patterns risk disrupting legitimate users.

Current research on API security risk assessment using dynamic ML models [14] demonstrates that machine learning can optimize rate limits by predicting whether request patterns indicate legitimate use or abuse. The work by Nokovic et al. [14] shows that ML models trained on authentication attempt patterns can significantly reduce false access probability even when dealing with imbalanced datasets typical of authentication systems. These systems adapt limits based on user history, request characteristics, and system load, maintaining security without unnecessarily restricting legitimate usage. Request characteristics include API endpoint patterns, payload sizes, and request timing. Their approach combines qualitative policy rules with quantitative risk scores, enabling fine-grained rate limiting decisions that traditional static threshold systems cannot achieve.

3.5.2 API Gateway Security. API gateways sit between ML prediction services and external clients, providing centralized enforcement of security policies: authentication, authorization, rate limiting, input validation, and logging. Gateways enable security policy changes without modifying model serving code—critical for ML systems where data scientists may lack security expertise.

However, API gateways introduce potential bottlenecks and single points of failure. Gateway performance directly impacts end-to-end inference latency, and gateway downtime blocks all prediction requests. Distributed gateway architectures with multiple instances behind load balancers provide redundancy, but introduce configuration complexity and potential policy inconsistencies across instances.

3.5.3 Request Validation and Sanitization. ML APIs must validate requests before processing to prevent injection attacks, malformed inputs that crash services, or adversarial examples designed to manipulate predictions. Validation includes schema checking (correct field types and required fields), range checking (values within expected bounds), and semantic validation (inputs match expected distributions).

However, overly strict validation can reject legitimate but unusual inputs—precisely the scenario where ML models should provide predictions. Research on API security for ML systems shows that ML-based input validation adapts to changing input distributions while detecting truly anomalous inputs likely to represent attacks. These systems learn normal input characteristics from traffic patterns, flagging inputs that deviate significantly while accommodating gradual distribution shifts.

3.6 Summary and Integration

Access control for ML deployments requires integrating multiple mechanisms—authentication, authorization, multi-tenant isolation, IAM, and API security—into defense-in-depth architectures. No single mechanism provides complete protection; rather, layered controls create security resilience where failures in one layer are caught by others.

Key challenges remain in bridging research and practice. Many advanced access control techniques (ABAC policy verification, behavioral biometrics, ML-based anomaly detection) show promise in research settings but lack production-ready implementations. Organizations need practical guidance on:

selecting appropriate authentication mechanisms for different ML workload types, designing RBAC/ABAC policies that balance security and operational efficiency, implementing multi-tenant isolation that withstands sophisticated attacks, managing service account proliferation in complex ML pipelines, and configuring API security that prevents abuse without degrading user experience.

The following sections examine runtime security mechanisms that complement access control, providing continuous protection during ML system operation, and operational best practices that sustain security postures over time.

4 Runtime Security

While access control prevents unauthorized access to ML systems, runtime security mechanisms provide continuous protection during system operation. This section examines techniques for validating inputs, monitoring system behavior, detecting anomalies, identifying threats, managing request rates, and maintaining comprehensive audit logs. Runtime security must operate with minimal performance overhead while detecting sophisticated attacks that evade static defenses.

4.1 Input Validation and Adversarial Detection

ML models process diverse inputs—images, text, sensor data, structured features—each requiring validation to prevent malicious manipulation. Input validation operates at multiple levels: syntactic validation ensures inputs conform to expected schemas, semantic validation checks that inputs represent plausible real-world data, and adversarial detection identifies inputs crafted to manipulate model predictions.

4.1.1 Schema and Type Validation. Schema validation forms the first line of runtime defense, rejecting requests that violate API contracts. For ML inference APIs, this includes verifying that input tensors have correct dimensions, feature values match expected types (numerical, categorical, text), required fields are present, and value ranges fall within trained bounds. Schema validation catches implementation errors, malformed requests from buggy clients, and naive attack attempts.

However, schema validation alone provides insufficient protection for ML systems. Adversarial examples—inputs specifically crafted to cause misclassification—typically satisfy all schema constraints while exploiting model vulnerabilities. A slightly perturbed image that passes all format checks might cause an autonomous vehicle to misclassify a stop sign, or a carefully crafted text input might extract sensitive information from a language model.

4.1.2 Adversarial Input Detection. Detecting adversarial inputs requires analyzing input characteristics beyond schema compliance. Research on security and privacy in machine learning [27] provides a comprehensive threat model categorizing attacks on ML systems. Papernot et al. [27] systematize findings on ML security, identifying that adversarial examples exploit the high-dimensional input space and model decision boundaries. Their framework shows that adversarial detection must account for both white-box attacks (where adversaries know model architecture) and black-box attacks (where adversaries only access predictions).

Defense approaches include input preprocessing (transforming inputs to remove adversarial perturbations), ensemble methods (using multiple models with different architectures), and statistical analysis (detecting inputs that deviate from training distributions). However, these defenses introduce tradeoffs: preprocessing may degrade legitimate inputs, ensemble methods increase inference latency and cost, and statistical detection generates false positives on unusual but legitimate inputs.

4.1.3 Distribution Shift and Data Drift Detection. Beyond adversarial attacks, ML models face gradual distribution shifts where real-world inputs diverge from training data. Concept drift—changes in the relationship between inputs and outputs—degrades model accuracy over time. Runtime monitoring must detect drift to trigger model retraining or alert operators to degraded performance.

Statistical methods track input feature distributions, comparing recent inference requests against training data statistics. Significant deviations indicate drift requiring investigation. However, distinguishing malicious distribution shifts (attacks) from benign shifts (changing user behavior) remains challenging. ML-based drift detection adapts thresholds based on historical patterns, reducing false alarms while maintaining attack detection.

4.2 Anomaly Detection in ML Systems

Anomaly detection identifies unusual patterns in system behavior that may indicate attacks, failures, or misconfigurations. For ML deployments, anomaly detection operates at multiple levels: network traffic patterns, API request characteristics, model prediction distributions, and infrastructure resource usage.

4.2.1 Foundations of ML-Based Anomaly Detection. The systematic review by Bou Nassif et al. [24] provides comprehensive analysis of machine learning for anomaly detection, examining 290 research articles from 2000-2020. Their work identifies 43 different applications of anomaly detection and 29 distinct ML models used for identifying anomalies. Key findings reveal that unsupervised anomaly detection methods (clustering, autoencoders, isolation forests) dominate research because labeled anomaly data is scarce—most systems operate normally, with attacks representing rare events.

For ML deployment security, this scarcity of attack data creates challenges: anomaly detectors trained only on normal behavior may generate high false positive rates when encountering legitimate but unusual patterns. The review identifies

that supervised methods achieve better accuracy when attack examples are available, but require continuous updating as attack patterns evolve. Semi-supervised approaches—training on normal data with limited attack examples—provide middle ground, though their effectiveness depends on attack diversity in training data.

4.2.2 Network Anomaly Detection. Wang et al. [25] survey machine learning in network anomaly detection across traditional networks, software-defined networks (SDN), Internet of Things (IoT), and cloud environments. Their comprehensive analysis shows that ML-based intrusion detection systems outperform signature-based approaches for detecting novel attacks, particularly zero-day exploits not present in signature databases.

For ML infrastructure deployed in cloud or on-premises data centers, network anomaly detection must handle: high-volume legitimate traffic from inference requests, distributed architectures where model serving spans multiple services, encrypted traffic that prevents deep packet inspection, and multi-tenant scenarios where traffic from different organizations shares network infrastructure. The survey identifies that feature engineering—selecting appropriate network characteristics for ML models—critically impacts detection accuracy. Effective features include flow statistics (packet counts, byte counts, duration), temporal patterns (inter-arrival times, burstiness), and connection characteristics (source/destination ports, protocol types).

4.2.3 Real-Time Anomaly Detection Architecture. Zhao et al. [26] present a novel framework for real-time network traffic anomaly detection using machine learning at scale. Their system, deployed on the University of Missouri–Kansas City campus network, combines Apache Kafka for distributed message streaming, Apache Storm for real-time computation, and machine learning algorithms for anomaly classification. This architecture processes network flow data in real-time, identifying anomaly patterns while handling massive data volumes.

Key architectural insights for ML deployment monitoring include: stream processing frameworks enable real-time analysis without blocking traffic, distributed computation scales horizontally as monitoring load increases, and selective analysis (applying expensive ML models only to suspicious traffic) balances detection accuracy with computational cost. Their preliminary results demonstrate feasibility of real-time ML-based monitoring for production networks, though challenges remain in tuning detection thresholds, handling concept drift in traffic patterns, and minimizing false positives that generate alert fatigue.

4.3 Threat Detection and Classification

While anomaly detection identifies unusual patterns, threat detection classifies specific attack types and assesses severity to prioritize responses.

4.3.1 AI-Enhanced Threat Detection. Research on AI and cyber security by Katiyar et al. [28] examines how machine learning enhances threat detection and response. Their work demonstrates that AI techniques enable more effective and efficient threat detection compared to traditional signature-based approaches, particularly for sophisticated and evolving cyber threats. Key ML algorithms for threat detection include: supervised learning for malware classification (identifying malicious software from features), deep learning for network intrusion detection (analyzing packet sequences), and reinforcement learning for adaptive defense strategies (learning optimal responses to different attack types).

For ML deployment security, threat detection must identify attacks specific to ML systems: adversarial example generation, model extraction through API abuse, membership inference attacks, and data poisoning attempts. Each attack type exhibits distinct signatures in request patterns, prediction outputs, or system resource usage that ML-based threat detection can learn to recognize.

4.3.2 Side-Channel Attack Detection. ML systems in shared infrastructure face side-channel attacks where adversaries infer sensitive information from observable system behavior. Research on machine learning for side-channel attack detection at runtime [29] shows that ML techniques can identify subtle patterns in system telemetry indicating information leakage through timing variations, cache access patterns, or power consumption.

However, side-channel detection introduces significant overhead: collecting fine-grained telemetry impacts performance, analyzing high-dimensional time-series data requires substantial computation, and distinguishing side-channel attacks from normal performance variations generates false positives. Production deployments must carefully balance security monitoring against performance degradation, potentially applying intensive monitoring only to high-value or high-risk workloads.

4.4 Logging and Audit Trails

Comprehensive logging provides forensic evidence for security investigations, compliance auditing, and system debugging. ML systems must log authentication attempts, authorization decisions, prediction requests and responses, model loading and deployment events, and security policy changes.

4.4.1 Security-Relevant Event Logging. Effective security logging captures: who accessed what resources, when access occurred, from where requests originated, what operations were performed, whether access was granted or denied, and what data was accessed or modified. For ML systems, additional events require logging: which model versions served predictions, what input features were provided, what predictions were returned, and what confidence scores accompanied predictions.

However, logging ML inference at scale generates massive data volumes. High-throughput ML services processing thousands of predictions per second cannot afford to log every request in full detail. Selective logging strategies capture all authentication and authorization events, sample routine inference requests, and log all suspicious or high-risk predictions. This balances forensic capability with storage costs and logging overhead.

4.4.2 Secure Log Storage and Analysis. Logs themselves become security-critical assets requiring protection. Adversaries who compromise systems often attempt to delete or modify logs to hide their activities. Secure logging requires: tamper-evident storage (append-only logs or blockchain-based logging), access controls restricting log viewing to authorized personnel, encryption protecting sensitive information in logs, and retention policies balancing forensic needs with storage costs and privacy regulations.

Log analysis for security monitoring must process high-volume log streams in real-time, correlating events across distributed systems to identify attack patterns. Machine learning techniques enable automated log analysis, detecting anomalous sequences of events that indicate attacks. However, ML-based log analysis faces the cold-start problem: models require training data representing both normal operations and attacks, yet sophisticated attacks may not appear in historical logs.

4.5 Rate Limiting and Resource Management

Rate limiting and resource quotas prevent resource exhaustion attacks while ensuring fair sharing among tenants. For ML systems, these mechanisms serve dual purposes: protecting infrastructure availability and preventing information leakage through repeated queries.

4.5.1 Request Rate Limiting. Request rate limiting restricts the number of API calls users or services can make within time windows. For ML inference APIs, rate limits prevent: denial-of-service attacks exhausting compute resources, model extraction attacks requiring many queries to reconstruct models, cost overruns from excessive inference requests, and unfair resource consumption by individual tenants in multi-tenant deployments.

Static rate limits (e.g., 1000 requests per hour) provide simple implementation but fail to accommodate varying legitimate usage patterns. Dynamic rate limiting adapts to context: trusted users receive higher limits, suspicious behavior triggers tighter restrictions, and system load influences available capacity. However, dynamic limits introduce complexity: determining appropriate thresholds, avoiding feedback loops where legitimate users are incorrectly restricted, and ensuring fairness across tenants with different usage patterns.

4.5.2 Resource Quotas and Isolation. Beyond request counts, resource quotas limit compute, memory, and storage consumption. Kubernetes resource quotas prevent individual tenants from monopolizing cluster resources. For ML workloads, quotas must account for: inference latency requirements (real-time predictions need guaranteed resources), batch processing patterns (periodic high-volume requests), model size variations (large language models require more memory than small classifiers), and GPU sharing (multiple tenants sharing expensive accelerators).

However, static resource quotas create inefficiencies: resources sit idle when tenants underutilize allocations, while tenants face artificial constraints during legitimate usage spikes. Elastic quotas that expand during low cluster utilization and contract during contention provide better resource efficiency, but introduce fairness concerns and potential gaming where tenants time requests to exploit elastic policies.

4.6 Summary and Integration

Runtime security for ML deployments requires continuous monitoring and adaptive defenses operating throughout system execution. Input validation prevents malicious data from reaching models, output monitoring detects unusual prediction patterns, anomaly detection identifies deviations from normal behavior, threat detection recognizes specific attack types, and logging provides forensic evidence and compliance documentation.

Key research findings from our analysis of 22 papers on runtime security reveal several critical insights. First, unsupervised anomaly detection dominates approaches due to scarcity of labeled attack data [24], yet production systems require low false positive rates that unsupervised methods struggle to achieve. Second, real-time detection systems must balance detection accuracy with processing latency [26], creating fundamental tradeoffs between security and performance. Third, ML-based security monitoring itself introduces attack surfaces: adversaries can poison training data for anomaly detectors or craft attacks that evade learned detection models [27].

Organizations deploying ML systems need practical guidance on: selecting appropriate anomaly detection algorithms for different attack types and system characteristics, configuring detection thresholds that balance false positives and false negatives, implementing real-time monitoring architectures that scale with inference load, designing logging strategies that capture security-relevant events without overwhelming storage, and integrating runtime security with access control and operational practices into cohesive defense-in-depth strategies.

The following sections examine infrastructure security mechanisms and operational best practices that complement access control and runtime security.

5 Infrastructure Security

Infrastructure security encompasses the foundational layers supporting ML deployments: network architecture, container platforms, storage systems, and deployment pipelines. While access control and runtime security protect against unauthorized access and malicious behavior, infrastructure security

ensures the underlying platform resists attacks and maintains isolation between components. This section examines security mechanisms specific to ML infrastructure, drawing from our analysis of 22 papers on container security, network protection, and deployment architectures.

5.1 Container and Orchestration Security

Containerization has become the standard deployment model for ML systems, with Kubernetes dominating orchestration. However, containers introduce security challenges distinct from traditional virtualization, particularly for ML workloads with sensitive models and data.

5.1.1 Container Isolation Mechanisms. As established in our background section [3], containers share the host kernel while providing isolated execution environments through Linux kernel features: namespaces (isolating process IDs, network interfaces, file systems), control groups or cgroups (limiting resource consumption), capabilities (restricting privileged operations), and seccomp (filtering system calls). For ML deployments, proper container isolation prevents: model weights in one container from being accessed by other containers, GPU memory from leaking between tenants sharing accelerators, and compromised inference services from escalating to host access.

However, container isolation faces fundamental limitations. All containers share the kernel, creating a single point of failure: kernel vulnerabilities enable container escape attacks that compromise the entire host. Research demonstrates that cryptomining malware can exploit container misconfigurations to hijack cluster resources [4], highlighting the need for continuous monitoring beyond static isolation configuration. The emergence of 5G networks has accelerated container adoption for Virtual Network Functions (VNFs), making container security even more critical [35]. AI-driven approaches for container security in 5G environments show that containers provide improved scalability, flexibility, and efficiency for network functions, but their portability and on-demand deployment also expand the attack surface requiring advanced security mechanisms.

5.1.2 Kubernetes Security Best Practices. Kubernetes provides powerful orchestration capabilities but introduces complex security considerations. Security best practices for ML deployments include: running containers as non-root users to limit privilege escalation, using Pod Security Policies or Pod Security Standards to enforce security constraints, enabling network policies to restrict pod-to-pod communication, implementing resource quotas to prevent resource exhaustion, and regularly updating Kubernetes versions to patch security vulnerabilities.

For ML-specific scenarios, additional considerations apply: GPU-enabled nodes require special security attention due to limited GPU virtualization, model serving pods need secure volume mounts for model weights, and training workloads may require privileged access for performance optimization (creating security-performance tradeoffs). Organizations must balance Kubernetes security hardening with the operational flexibility ML teams require for experimentation and rapid iteration.

5.2 Network Security and Segmentation

Network security for ML infrastructure must protect multiple communication paths: external clients accessing prediction APIs, internal services communicating within ML pipelines, administrative access to infrastructure, and data movement between storage and compute.

5.2.1 Network Segmentation Strategies. Network segmentation divides infrastructure into isolated zones with controlled communication paths. For ML deployments, typical segmentation includes: public-facing zone for API gateways and load balancers, application zone for model serving services, data zone for training data and model storage, and management zone for administrative access and monitoring. Firewalls or network policies enforce rules restricting traffic between zones.

However, ML workflows require complex cross-zone communication: inference services must access model storage, training pipelines must read data and write models, and monitoring must collect metrics from all zones. Overly restrictive segmentation blocks legitimate workflows, while overly permissive rules create attack paths. Organizations need clear documentation of required communication patterns and regular audits verifying that network policies match intended architecture.

5.2.2 Service Mesh Security. Service meshes (Istio, Linkerd, Consul) provide infrastructure-level security for microservices communication. For ML deployments, service meshes offer: automatic mutual TLS between services, fine-grained traffic policies, distributed tracing for debugging, and centralized observability. These capabilities particularly benefit complex ML pipelines with multiple services (feature stores, model servers, caching layers, monitoring).

However, service meshes introduce operational complexity and performance overhead. Each request passes through sidecar proxies that handle encryption and policy enforcement, adding latency to inference paths. For latency-sensitive ML applications, this overhead may be unacceptable. Organizations must evaluate whether service mesh benefits justify complexity and performance costs for their specific ML deployment patterns.

5.3 Data Security and Model Protection

ML systems must protect multiple data types with different security requirements: training data (often sensitive or proprietary), model weights (intellectual property), inference

inputs (potentially containing PII), and prediction outputs (may reveal sensitive information).

5.3.1 Encryption at Rest and in Transit. Encryption protects data from unauthorized access during storage and transmission. For ML systems, encryption applies to: training datasets stored in object storage or databases, model weights stored in model registries, inference requests and responses transmitted over networks, and logs containing potentially sensitive information. Modern cloud platforms provide encryption by default, but organizations must manage encryption keys, ensure proper key rotation, and verify that encryption actually protects against their threat model.

However, encryption introduces performance overhead and operational complexity. Decrypting large training datasets impacts training pipeline performance, encrypted model loading adds latency to inference startup, and key management systems become critical dependencies. For GPU-accelerated ML workloads, data must be decrypted before GPU processing, creating windows where plaintext exists in memory. Trusted execution environments (TEEs) like Intel SGX or AMD SEV provide hardware-based encryption of memory contents, but with significant performance penalties and limited memory sizes unsuitable for large models.

5.3.2 Model Weight Protection. Model weights represent significant intellectual property investment and may encode sensitive information from training data. Protection mechanisms include: access controls restricting who can download models, encryption of stored model files, watermarking to trace leaked models, and API-only access preventing direct model extraction. For commercial ML services, model protection directly impacts business value: leaked models enable competitors to replicate capabilities without training costs.

However, models served through prediction APIs remain vulnerable to extraction attacks where adversaries reconstruct models through repeated queries. Rate limiting provides partial protection but cannot prevent determined attackers with sufficient time and resources. Research on model extraction defenses explores techniques like prediction perturbation and query auditing, but these introduce accuracy degradation or monitoring overhead. Organizations must assess model value against protection costs, potentially accepting extraction risk for low-value models while implementing stronger protection for high-value proprietary models.

5.4 Deployment Pipeline Security

ML deployment pipelines automate the process of moving models from development to production. Securing these pipelines prevents adversaries from injecting malicious models, tampering with deployment configurations, or compromising production systems through CI/CD vulnerabilities.

5.4.1 CI/CD Security for ML. Continuous integration and deployment (CI/CD) pipelines for ML extend traditional software CI/CD with ML-specific stages: data validation, model training, model evaluation, model packaging, and deployment. Each stage introduces security considerations: training code may contain vulnerabilities, model evaluation may use poisoned test data, and deployment configurations may expose secrets.

Security practices for ML CI/CD include: code review for training scripts, signed commits to prevent unauthorized changes, isolated build environments to prevent supply chain attacks, artifact signing to verify model authenticity, and staged deployments with rollback capabilities. However, ML teams often lack security expertise, and security teams lack ML knowledge, creating gaps in CI/CD security practices.

5.4.2 Model Registry Security. Model registries (MLflow, DVC, cloud-native solutions) store trained models with metadata, versioning, and lineage tracking. Registry security must prevent: unauthorized model downloads, model tampering or replacement, metadata manipulation, and access to sensitive training information. Access controls, encryption, and audit logging provide baseline protection, but registries become high-value targets as organizations centralize model storage.

5.5 Summary and Integration

Infrastructure security for ML deployments requires securing multiple layers: container platforms providing isolated execution, network architecture controlling communication, encryption protecting data and models, and deployment pipelines ensuring model integrity. Our analysis of 22 infrastructure security papers reveals that security failures often stem from misconfiguration rather than missing features: Kubernetes provides strong isolation mechanisms, but default configurations often leave them disabled; encryption is available, but key management complexity leads to weak implementations; network segmentation is possible, but complex ML workflows lead to overly permissive rules.

Organizations need practical guidance on: configuring Kubernetes security features without breaking ML workflows, implementing network segmentation that accommodates ML communication patterns, managing encryption keys at scale, protecting model intellectual property while enabling serving, and securing CI/CD pipelines with ML-specific considerations. The following section examines operational best practices that sustain infrastructure security over time.

6 Operational Best Practices

While previous sections examined technical security mechanisms, this section addresses operational practices that sustain security over time. Drawing from our analysis of 18 papers on MLOps, compliance, and security operations, we examine how organizations integrate security into ML workflows, maintain regulatory compliance, respond to incidents, and build security culture across data science, engineering, and operations teams.

6.1 MLSecOps: Integrating Security into ML Operations

MLSecOps extends MLOps practices by embedding security throughout the ML lifecycle. As discussed in our background section [5, 6], early MLOps focused on automation and reliability, with security added as an afterthought. MLSecOps addresses this by making security a first-class concern from development through deployment and maintenance.

6.1.1 Secure Development Practices. Secure ML development begins with secure data handling: validating data sources, sanitizing training data to remove malicious examples, and implementing access controls for sensitive datasets. Code security practices include: reviewing training scripts for vulnerabilities, scanning dependencies for known security issues, and using secure coding practices for inference services. Model security involves: validating model architectures for known vulnerabilities, testing models for adversarial robustness, and documenting model limitations and failure modes.

However, data science teams often lack security training, viewing security practices as obstacles to experimentation. Organizations must balance security requirements with the flexibility data scientists need for research and development. Automated security checks integrated into development workflows (security linters, dependency scanners, automated adversarial testing) provide security guardrails without blocking innovation.

6.1.2 Continuous Security Monitoring. MLSecOps requires continuous monitoring throughout the ML lifecycle: monitoring training for data poisoning attempts, monitoring model performance for drift or degradation, monitoring inference for adversarial attacks or abuse, and monitoring infrastructure for misconfigurations or vulnerabilities. This comprehensive monitoring generates massive telemetry requiring automated analysis.

As demonstrated in our runtime security section, machine learning techniques enable automated security monitoring [24, 25]. However, monitoring ML systems with ML creates recursive challenges: who monitors the monitoring systems? How do organizations prevent adversaries from poisoning monitoring models? These questions require careful architectural design with human oversight of critical security decisions.

6.2 Regulatory Compliance

ML systems processing personal or sensitive data must comply with regulations including GDPR (Europe), HIPAA (US healthcare), CCPA (California), and industry-specific requirements (PCI-DSS for payments, SOC 2 for service providers). Compliance requirements span data handling, model transparency, privacy protection, and accountability.

6.2.1 GDPR and Privacy Requirements. GDPR imposes requirements particularly challenging for ML systems [9]: data minimization (collecting only necessary data), purpose limitation (using data only for stated purposes), transparency (explaining how data is used), and individual rights (data access, deletion, portability). ML models trained on personal data must satisfy these requirements while maintaining utility.

Research on data minimization for GDPR compliance in machine learning models [38] addresses the fundamental tension between GDPR's requirement to collect only necessary data and ML algorithms' tendency to consume large amounts of data for predictions. Neural networks and other "black box" models make it difficult to derive exactly which data influenced decisions, complicating demonstration of data minimization compliance. Organizations must balance model accuracy (which often improves with more data) against regulatory requirements for minimal data collection.

Automated compliance checking using combined rule-based and machine learning approaches [36] shows promise for reducing manual compliance burden. These systems analyze privacy policies, data processing procedures, and system architectures to identify potential GDPR violations. AI frameworks to support GDPR compliance decisions [37], such as the INTREPID system for Italian Public Administration, demonstrate that ML can help organizations ensure document compliance. However, automated compliance checking cannot replace legal expertise: ML systems flag potential issues, but legal professionals must interpret results and make final compliance determinations.

Key compliance challenges include: right to explanation (explaining model predictions), right to deletion (removing individual data from trained models), and data protection impact assessments (quantifying privacy risks). Research on privacy-preserving ML [8] provides technical mechanisms (differential privacy, federated learning), but implementing these in production requires expertise most organizations lack. Compliance often relies on procedural controls (data governance policies, consent management) rather than technical privacy guarantees.

6.2.2 Model Governance and Auditability. Regulatory compliance requires comprehensive documentation of ML systems: what data was used for training, how models were evaluated, what decisions models make, and how model outputs are used. Model governance frameworks track model lineage, maintain model registries with metadata, implement approval workflows for production deployment, and provide audit trails for compliance verification.

However, model governance introduces overhead that slows ML development cycles. Organizations must balance compliance requirements with the rapid iteration that ML development requires. Automated governance tools integrated into MLOps pipelines can capture metadata and lineage without manual documentation, but require upfront investment in tooling and process design.

6.3 Incident Response and Recovery

Security incidents in ML systems require specialized response procedures beyond traditional incident response. ML-specific

incidents include: adversarial attacks causing systematic misclassifications, model extraction or theft, data poisoning requiring model retraining, and privacy breaches through model inversion or membership inference.

6.3.1 Incident Detection and Triage. Detecting ML-specific security incidents requires monitoring for unusual patterns: sudden accuracy degradation (potential adversarial attack or poisoning), unusual API access patterns (potential model extraction), or anomalous predictions (potential adversarial inputs). Automated alerting systems must distinguish security incidents from operational issues (bugs, infrastructure failures, data quality problems).

Incident triage determines severity and required response. Model extraction attempts may require immediate API access revocation, while adversarial attacks may need model rollback to previous versions. However, ML incident response often lacks clear playbooks: organizations may not know whether to retrain models, roll back to previous versions, or implement additional input validation. Building ML-specific incident response procedures requires collaboration between security teams (incident response expertise) and data science teams (ML system understanding).

6.3.2 Recovery and Post-Incident Analysis. Recovering from ML security incidents may require: retraining models on cleaned data (for poisoning attacks), rotating compromised credentials, patching vulnerabilities in inference services, or implementing additional security controls. Post-incident analysis must determine: how attackers gained access, what data or models were compromised, whether attacks succeeded in their objectives, and what controls would have prevented the incident.

For ML systems, post-incident analysis faces unique challenges: determining whether training data was poisoned requires analyzing potentially millions of examples, assessing whether models were extracted requires analyzing API access logs for suspicious patterns, and quantifying privacy breaches requires statistical analysis of model outputs. Organizations need tools and expertise for ML-specific forensics, which remain underdeveloped compared to traditional security forensics.

6.4 Security Training and Culture

Effective ML security requires collaboration across teams with different expertise: security teams understand threats and defenses but lack ML knowledge, data science teams understand models but lack security training, and operations teams manage infrastructure but may not recognize ML-specific vulnerabilities. Building security culture requires cross-functional training and shared responsibility.

6.4.1 Cross-Functional Security Training. Security training for ML teams should cover: common ML attack types (adversarial examples, model extraction, poisoning), secure coding practices for ML systems, privacy-preserving techniques, and incident response procedures. Conversely, ML training for security teams should cover: how ML models work and their limitations, ML deployment architectures, ML-specific attack surfaces, and performance constraints affecting security controls.

However, comprehensive cross-training requires significant time investment that organizations often cannot afford. Targeted training focused on specific roles and responsibilities provides more practical approach: data scientists learn secure model development, ML engineers learn secure deployment practices, and security teams learn ML threat models. Regular security reviews of ML systems provide learning opportunities and catch security issues before production deployment.

6.5 Summary and Integration

Operational best practices sustain security over time through processes, culture, and continuous improvement. Our analysis of 18 papers on MLOps, compliance, and security operations reveals that technical security mechanisms alone provide insufficient protection: organizations need operational practices that integrate security into ML workflows, maintain compliance with evolving regulations, respond effectively to incidents, and build security expertise across teams.

Key challenges include: bridging expertise gaps between security, data science, and operations teams; balancing security requirements with ML development velocity; implementing compliance requirements without overwhelming teams with documentation; and building incident response capabilities for ML-specific threats. Organizations succeeding at ML security treat it as a shared responsibility across functions, invest in cross-functional training, and automate security controls to reduce manual burden.

7 Taxonomy of Security Mechanisms

This section synthesizes the security mechanisms examined in previous sections into a unified taxonomy, providing a structured view of the security landscape for ML deployment. We organize mechanisms by their primary function, identify relationships between mechanisms, and compare approaches across multiple dimensions.

7.1 Taxonomy Structure

Our taxonomy organizes ML deployment security mechanisms along three primary dimensions corresponding to the survey's core pillars (Figure 1):

Access Control Mechanisms (Section 3) govern who can access ML systems and what operations they can perform. This dimension includes authentication (verifying identity), authorization (granting permissions), multi-tenant isolation (separating tenants), identity management (managing credentials), and API security (protecting interfaces). These mechanisms operate primarily at system boundaries, controlling entry to ML infrastructure and services.

Runtime Security Mechanisms (Section 4) provide continuous protection during system operation. This dimension includes input validation (checking request validity), anomaly detection (identifying unusual patterns), threat detection (recognizing specific attacks), logging (recording events), and rate limiting (controlling resource consumption). These mechanisms operate continuously, monitoring system behavior and responding to threats.

Infrastructure Security Mechanisms (Section 5) protect the underlying platform supporting ML deployments. This dimension includes container isolation (separating workloads), network segmentation (controlling communication), encryption (protecting data), deployment security (securing pipelines), and model protection (safeguarding intellectual property). These mechanisms operate at infrastructure layers, providing foundational security properties.

Operational Practices (Section 6) sustain security over time through processes and culture. This dimension includes MLSecOps (integrating security into workflows), compliance (meeting regulations), incident response (handling security events), and training (building expertise). These practices operate at organizational level, ensuring security mechanisms are properly configured, maintained, and improved.

7.2 Mechanism Relationships and Dependencies

Security mechanisms do not operate independently; rather, they form an integrated defense-in-depth architecture where mechanisms complement and reinforce each other:

Access Control enables Runtime Security: Authentication and authorization determine which entities can access ML systems, while runtime monitoring detects abuse of authorized access. For example, API authentication grants access to prediction services, while rate limiting and anomaly detection prevent authorized users from extracting models through excessive queries.

Infrastructure Security supports Access Control: Container isolation and network segmentation provide the foundation for multi-tenant access control. Without proper infrastructure isolation, access control policies cannot prevent cross-tenant information leakage through side channels or shared resources.

Runtime Security informs Access Control: Anomaly detection and threat detection identify patterns indicating compromised credentials or policy violations, triggering access revocation or additional authentication requirements. This feedback loop enables adaptive security that responds to observed threats.

Operational Practices sustain all mechanisms: MLSecOps practices ensure security mechanisms are properly configured, compliance requirements drive security control selection, incident response procedures leverage logs and monitoring, and training builds expertise to implement mechanisms effectively.

ML Deployment Security Taxonomy

1. ACCESS CONTROL (Section 3) – *Who can access ML systems?*
- Authentication: API Keys, OAuth/JWT, mTLS, MFA
- Authorization: RBAC, ABAC, Dynamic Policies
- Multi-Tenancy: Namespace Isolation, Data Encryption
- Identity Management: IAM, SSO, Federation
- API Security: Rate Limiting, Input Validation

2. RUNTIME SECURITY (Section 4) – *Detecting & preventing attacks*
- Input Validation: Schema Checks, Sanitization
- Anomaly Detection: Statistical, ML-Based
- Threat Detection: Adversarial, Model Extraction
- Logging & Monitoring: Audit Logs, Metrics
- Response: Rate Limiting, Request Blocking

3. INFRASTRUCTURE SECURITY (Section 5) – *Platform protection*
- Container Security: Isolation, Sandboxing (gVisor)
- Orchestration: Kubernetes RBAC, NetworkPolicies
- Network Security: Segmentation, Service Mesh
- Data Protection: Encryption, Key Management
- Deployment: Secure CI/CD, Model Registry

4. OPERATIONAL PRACTICES (Section 6) – *Sustaining security*
- MLSecOps: Security Integration, Automation
- Compliance: GDPR, HIPAA, Privacy Impact
- Incident Response: Detect, Contain, Recover
- Culture: Training, Cross-functional Teams

Integration Principle: *Defense-in-Depth requires mechanisms from all four pillars working together. No single layer provides complete protection.*

Fig. 1. Hierarchical taxonomy of security mechanisms for ML deployment infrastructure. The taxonomy organizes 45+ security mechanisms into four pillars by primary function. Each pillar addresses distinct aspects of the security lifecycle and must integrate with others for comprehensive defense-in-depth protection.

7.3 Comparative Analysis

We compare security mechanisms across multiple dimensions relevant to ML deployment practitioners:

Security Properties: Different mechanisms provide different security guarantees. Authentication prevents impersonation, authorization prevents unauthorized operations, encryption prevents data disclosure, and anomaly detection identifies attacks. No single mechanism provides complete protection; defense-in-depth requires multiple complementary mechanisms.

Performance Impact: Security mechanisms introduce varying performance overhead. Authentication adds latency to each request (milliseconds for API keys, more for OAuth

flows), encryption impacts data transfer and storage performance, and continuous monitoring consumes CPU and memory. Organizations must balance security requirements with performance constraints, potentially accepting weaker security for latency-critical applications.

Operational Complexity: Mechanisms vary in deployment and maintenance complexity. API key authentication is simple to implement but challenging to manage at scale, while certificate-based authentication requires PKI infrastructure but provides stronger security. Service meshes automate mTLS but introduce operational complexity. Organizations must assess whether their teams have expertise to operate complex security mechanisms.

Maturity and Tool Support: Mechanisms differ in production readiness. Container isolation and network policies have mature implementations in Kubernetes, while adversarial input detection remains largely research prototypes. Organizations should prioritize well-supported mechanisms for critical security requirements, using experimental techniques only where mature alternatives don't exist.

8 Discussion

This section synthesizes findings from our comprehensive analysis of ML deployment security, identifies critical gaps between research and practice, discusses emerging threats, and proposes directions for future research and development.

8.1 Key Findings and Insights

Examining 68 papers across access control, runtime security, infrastructure, and operations reveals several overarching insights about the current state of ML deployment security.

8.1.1 The Configuration-Over-Features Problem. A striking pattern emerges across all security dimensions: security failures stem far more often from misconfiguration than from missing features. Kubernetes provides robust isolation mechanisms—namespaces, network policies, RBAC—yet default configurations often leave them disabled or overly permissive. Cloud platforms offer encryption by default, but organizations struggle with key management complexities. Service meshes can automate mTLS, but teams often lack the expertise to configure them correctly.

This configuration-over-features problem reflects deeper organizational challenges. Security features designed by experts for experts get deployed by teams whose primary focus is ML model accuracy and system reliability. Documentation assumes security knowledge that ML teams typically lack, while security teams often miss the ML context needed to provide appropriate guidance. Solving this requires either simpler security mechanisms with secure defaults built in, or better integration of security expertise directly into ML teams.

8.1.2 The Performance-Security Tradeoff. Every security mechanism examined introduces some performance overhead, creating tension with ML systems' stringent latency and throughput requirements. Authentication adds milliseconds to each request, encryption slows data transfer rates, monitoring consumes CPU resources, and adversarial detection increases inference latency. For truly latency-sensitive applications—real-time recommendations, autonomous vehicles, high-frequency trading—even small overheads can prove unacceptable.

Yet the analysis reveals something encouraging: performance impact varies dramatically based on implementation quality. Well-optimized authentication using connection pooling and token caching adds negligible overhead, while naive implementations can double request latency. This suggests that organizations need guidance on implementing security mechanisms efficiently, not just binary advice about whether to implement them. Future research should optimize security mechanisms for ML workload characteristics rather than treating performance overhead as an inherent, unchangeable property.

Table 6 quantifies typical performance overhead introduced by different security mechanisms based on reported results across the surveyed papers. These values provide practitioners with guidance for estimating the performance impact of different security configurations. Actual overhead varies based on implementation quality, workload characteristics, and hardware configuration.

The table reveals that authentication and basic validation impose minimal overhead (¡5%), making them essential baseline security controls. More sophisticated mechanisms like adversarial detection (15-30%) and gVisor sandboxing (10-20%) require careful consideration of performance-security tradeoffs. Organizations should implement low-overhead mechanisms universally while selectively deploying high-overhead mechanisms for high-risk workloads. Notably, precision reduction (FP16, INT8) improves both performance and security posture by reducing attack surface, though with potential accuracy implications requiring validation.

8.1.3 The Research-Practice Gap. Many advanced security techniques remain confined to research papers without production implementations. Adversarial input detection, privacy-preserving training, and ML-based policy verification show promise in academic settings but lack mature tools, clear deployment guidance, or evidence of production viability. This gap reflects multiple factors: research focuses on novel techniques rather than engineering mature systems, academic threat models may not match production scenarios, and performance constraints in production exceed research assumptions.

Bridging this gap requires collaboration between researchers and practitioners: researchers need access to production workloads and constraints to validate techniques, while practitioners need research community engagement to articulate real-world security requirements. Industry-academic partnerships, open-source security tool development, and shared datasets representing production ML security scenarios could accelerate research translation.

Table 6. Performance Overhead of Security Mechanisms

Security Mechanism	Latency Overhead	Throughput Impact	References
Access Control			
API Key Authentication	¡1%	Minimal	[14]
OAuth Token Validation	2-5%	¡5%	[32]
mTLS Handshake	5-10% (first)	Minimal (cached)	[3]
Runtime Security			
Schema Validation	1-3%	¡2%	[10]
Adversarial Detection	15-30%	10-25%	[27]
Anomaly Detection	5-15%	5-10%	[24]
Comprehensive Logging	3-8%	¡5%	[13]
Infrastructure			
Container Isolation (Docker)	2-5%	¡3%	[3]
gVisor Sandboxing	10-20%	5-15%	[3]
Data Encryption (TLS)	5-15%	3-10%	[33]
Service Mesh (mTLS)	8-15%	5-12%	[3, 34]
Precision Impact			
FP32 (No Optimization)	Baseline	Baseline	[12]
FP16 (Security: Med)	0% (2x faster)	2x improvement	[12]
INT8 (Security: Lower)	0% (4x faster)	4x improvement	[12]

8.2 Current Limitations

Despite significant research progress, ML deployment security faces persistent limitations that our survey identifies:

Limited Adversarial Robustness: Defenses against adversarial examples remain fragile. Most defense techniques are subsequently broken by stronger attacks, creating an arms race without clear winners. Production ML systems lack reliable adversarial detection, forcing organizations to accept risk or avoid deploying models in adversarial environments.

Insufficient Privacy Guarantees: Privacy-preserving ML techniques (differential privacy, federated learning) provide theoretical guarantees but with substantial accuracy degradation or computational cost. Organizations needing strong privacy often cannot deploy these techniques in production, relying instead on procedural controls with weaker guarantees.

Immature Tooling: While container orchestration and CI/CD have mature tool ecosystems, ML-specific security tooling remains underdeveloped. Organizations lack tools for: automated adversarial testing, ML-specific vulnerability scanning, model extraction detection, privacy risk quantification, and ML security forensics.

Expertise Scarcity: ML security requires expertise spanning machine learning, systems security, and operations—a rare combination. Organizations struggle to hire or develop personnel with necessary breadth, creating security gaps where teams lack knowledge to implement appropriate controls.

8.3 Emerging Threats and Future Challenges

The ML deployment landscape continues evolving, introducing new security challenges:

Large Language Models (LLMs): The emergence of large language models with billions of parameters creates new security challenges: massive model sizes complicate secure storage and transfer, prompt injection attacks manipulate model behavior through crafted inputs, and models trained on internet data may memorize and leak sensitive information. Existing security mechanisms designed for smaller models may not scale to LLM deployments.

Edge and Federated Deployment: ML deployment increasingly occurs at edge devices and through federated learning across distributed data sources. These scenarios introduce security challenges beyond centralized cloud deployment: edge devices have limited security capabilities, federated learning exposes models to poisoning from compromised participants, and distributed deployments complicate security monitoring and incident response.

AI-Generated Code and Models: Tools generating code and models from natural language descriptions (GitHub Copilot, ChatGPT) introduce supply chain security risks: generated code may contain vulnerabilities, generated models may have backdoors, and organizations may lack visibility into how generated artifacts were created. Security practices must evolve to address AI-generated components in ML systems.

Quantum Computing Threats: Future quantum computers threaten current cryptographic protections. ML systems relying on encryption for data protection, certificate-based authentication, or secure communication must prepare for post-quantum cryptography migration. This transition will require updating encryption libraries, rotating keys, and potentially redesigning security architectures.

8.4 Decision Framework for Security Architecture Design

Organizations deploying ML systems face numerous security mechanism choices. This section provides decision frameworks and maturity models to guide security architecture design based on organizational context, threat profile, and operational constraints.

8.4.1 Security Maturity Model. We propose a four-level maturity model for ML deployment security, enabling organizations to progressively enhance protection as expertise and resources grow:

Level 1 - Minimal (Entry): Suitable for development environments, non-sensitive workloads, or organizations beginning ML security journey. Focus on essential controls with low operational overhead.

Authentication: API keys for service-to-service, basic password authentication for users.

Authorization: Simple RBAC with organizational roles (data scientist, ML engineer, admin).

Infrastructure: Standard container isolation with default Docker and Kubernetes settings, basic network segmentation with public and private subnets.

Runtime: Schema validation for inputs, basic logging to stdout/files.

Operational: Manual security reviews before production deployment, basic compliance documentation.

Overhead: ¡5% performance impact, minimal configuration complexity.

Suitable For: Internal development, proof-of-concept systems, low-risk applications.

Level 2 - Baseline (Production-Ready): Minimum recommended for production ML systems serving real users. Balances security and operational complexity.

Authentication: OAuth 2.0/JWT for user-facing APIs, API keys with rotation for services, MFA for administrative access.

Authorization: RBAC for organizational access, beginning ABAC policies for sensitive resources (model access, training data).

Infrastructure: Network policies restricting pod-to-pod communication, encrypted data at rest and in transit (TLS), regular security patching.

Runtime: Input validation with type/range checking, basic anomaly detection (statistical), comprehensive logging to centralized system, rate limiting by user/tenant.

Operational: Automated security scanning in CI/CD, regular security audits, incident response procedures documented.

Overhead: 5-10% performance impact, moderate configuration complexity.

Suitable For: Production systems with moderate risk, general enterprise ML applications, customer-facing services.

Level 3 - Enhanced (High-Security): For sensitive applications requiring strong protection. Implements advanced mechanisms with careful performance optimization.

Authentication: mTLS for service-to-service communication, OAuth with short-lived tokens, behavioral biometrics for continuous authentication (high-risk operations).

Authorization: Hybrid RBAC/ABAC with context-aware policies, ML-based anomaly detection for access patterns, fine-grained permissions (per-model, per-dataset).

Infrastructure: Service mesh (Istio/Linkerd) with automatic mTLS, enhanced container isolation (Pod Security Standards), network microsegmentation, encrypted secrets management (Vault).

Runtime: ML-based anomaly detection, lightweight adversarial input detection, real-time monitoring dashboards, automated threat response (rate limit escalation, account suspension).

Operational: MLSecOps with security integrated into all pipeline stages, automated compliance reporting, regular adversarial testing, dedicated security team for ML.

Overhead: 10-15% performance impact, high configuration complexity requiring specialized expertise.

Suitable For: Healthcare ML systems (HIPAA), financial services (fraud detection, trading), regulated industries, high-value models.

Level 4 - Maximum (Defense-in-Depth): For highest-security scenarios where security outweighs performance concerns. Implements all available mechanisms with multiple redundant layers.

Authentication: Hardware security keys, continuous authentication with behavioral analytics, certificate pinning.

Authorization: ML-based dynamic authorization adapting to threat level, zero-trust architecture (verify every access), real-time policy updates based on threat intelligence.

Infrastructure: gVisor/Kata containers for strong isolation, confidential computing (SGX/SEV) for sensitive data, air-gapped environments for model training, hardware security modules for key management.

Runtime: Comprehensive adversarial detection (accepting performance cost), ensemble anomaly detection (multiple algorithms), full request/response logging with retention, canary deployments for security testing.

Operational: Red team exercises, formal security verification, dedicated security operations center (SOC), real-time threat intelligence integration.

Overhead: 20-30% performance impact, very high complexity requiring security specialists.

Suitable For: Military/government ML, critical infrastructure, extremely high-value models, adversarial environments.

8.4.2 Decision Frameworks for Mechanism Selection. Selecting appropriate security mechanisms requires evaluating multiple factors including interaction model, security sensitivity, performance constraints, and operational requirements. Tables 7 and 8 provide decision criteria for authentication and authorization mechanisms, while Table 9 guides container isolation selection. These frameworks help practitioners match security mechanisms to deployment contexts based on evidence from the surveyed literature.

Table 7. Authentication Mechanism Selection Criteria

Interaction Type	Sensitivity	Recommended Mechanism	Section
Service-to-Service Communication			
Service	Low	API keys with rotation	3.1.1
Service	Medium	API keys + IP whitelist	3.1.1
Service	High	mTLS certificates	3.1.3
User-Facing Services			
User	Low	API keys or basic OAuth	3.1.2
User	Medium	OAuth 2.0 with JWT	3.1.2, 3.1.4
User	High	OAuth + MFA	3.1.2, 3.1.4
User	Critical	OAuth + MFA + biometrics	3.1.2, 3.1.5

Table 8. Authorization Model Selection Criteria

Complexity	Context-Aware	Recommended Model	Section
¡10 roles	No	RBAC	3.2.1
¡10 roles	Yes	RBAC + monitoring	3.2.1, 3.2.3
¿10 roles	No	RBAC with hierarchy	3.2.1
¿10 roles	Yes, contextual	ABAC	3.2.2
Complex rules	Yes, pattern detection	ABAC + ML monitoring	3.2.2, 3.2.3

Table 9. Container Isolation Mechanism Selection Criteria

Workload Type	Performance Tolerance	Recommended Isolation	Section
Trusted, single-tenant	Any	Standard Docker/K8s	5.1.1
Untrusted/multi-tenant	Low (¡5% overhead)	Standard + monitoring	5.1.1
Untrusted/multi-tenant	Medium (10-20%)	gVisor (syscall filtering)	5.1.1
Untrusted/multi-tenant	High (¿20% OK)	Kata Containers (VM-based)	5.1.1
Critical security	Any	Kata Containers + TEE	5.3.2

These selection criteria synthesize recommendations from Sections 3 (Access Control), 4 (Runtime Security), and 5 (Infrastructure Security), providing practitioners with evidence-based guidance for mechanism selection. Organizations should adapt these criteria to their specific threat models, regulatory requirements, and operational capabilities.

8.4.3 Security Configuration by Deployment Context. Table 10 provides recommended security configurations for common ML deployment scenarios. These recommendations balance security requirements with practical constraints for each context.

8.4.4 Trade-off Analysis and Decision Criteria. When selecting security mechanisms, organizations must explicitly evaluate trade-offs:

Security vs. Performance: Use Table 3 (Section 8.1.2) to estimate overhead. Prioritize low-overhead mechanisms (¡5%) universally. Deploy high-overhead mechanisms (15-30%) selectively for high-risk workloads or when threat model demands. Consider performance budgets: if overall security overhead must stay under 10%, choose 2-3 mechanisms at 3-5% each rather than one at 15%.

Security vs. Complexity: Simpler mechanisms (API keys, RBAC) reduce misconfiguration risk but provide limited protection. Complex mechanisms (ABAC, service mesh, behavioral biometrics) offer stronger security but require expertise to deploy correctly. Start simple and add complexity incrementally as organizational security maturity grows.

Security vs. Development Velocity: Strict security controls can slow ML experimentation. Consider tiered environments: permissive security in development (Level 1), production-ready security in staging (Level 2), full security in production (Level 3-4). Use automation (CI/CD security scanning, policy-as-code) to maintain velocity while enforcing security.

Security vs. Cost: Security mechanisms introduce costs: infrastructure (additional nodes for isolation, redundancy), tooling (commercial security products, monitoring platforms), and personnel (security engineers, security training). Quantify risk reduction value against costs. Prioritize high-impact, low-cost mechanisms first (authentication, basic RBAC, TLS encryption), then add higher-cost mechanisms as risk justifies.

This decision framework enables organizations to design security architectures matched to their specific contexts, constraints, and threat profiles. No single configuration suits all scenarios; successful ML security requires thoughtful mechanism selection based on explicit trade-off analysis.

8.5 Research Opportunities and Future Directions

Based on gaps identified throughout this survey, we propose several high-priority research directions:

Efficient Adversarial Defenses: Research should focus on adversarial detection and mitigation techniques with minimal performance overhead. Current defenses often double inference latency or degrade accuracy significantly. Techniques that provide reasonable security with acceptable performance tradeoffs would enable broader production deployment.

Automated Security Configuration: Tools that automatically configure security mechanisms based on ML workload characteristics and organizational requirements could address the configuration-over-features problem. Such tools would analyze ML deployment patterns, recommend appropriate security controls, and generate configurations with secure defaults.

ML-Specific Security Tooling: The ML security ecosystem needs mature tools for: continuous adversarial testing integrated into CI/CD, automated privacy risk assessment for trained models, model extraction detection and prevention, ML-specific vulnerability scanners, and security forensics for ML incidents. Open-source tool development would accelerate adoption.

Table 10. Recommended Security Configurations by Deployment Context

Context	Authentication	Authorization	Key Concerns
Development	API keys	Simple RBAC	Speed of iteration, minimal overhead
Staging	OAuth + API keys	RBAC	Production parity, testing security configs
Production (Internal)	OAuth/mTLS	RBAC + selective ABAC	Balance security and performance
Production (External)	OAuth + MFA	ABAC with rate limiting	API abuse prevention, strong auth
Regulated (Healthcare)	OAuth + MFA	ABAC + audit logs	HIPAA compliance, privacy, audit trails
Regulated (Financial)	mTLS + MFA	ABAC + real-time monitoring	Adversarial robustness, fraud detection
Multi-Tenant SaaS	OAuth per tenant	ABAC + strict isolation	Tenant separation, cost tracking
Edge Devices	Pre-shared keys	Minimal (device-level)	Resource constraints, offline operation

Privacy-Utility Tradeoff Optimization: Research should explore techniques that provide stronger privacy guarantees with less accuracy degradation. This includes: more efficient differential privacy mechanisms, federated learning with better convergence properties, and hybrid approaches combining technical and procedural privacy controls.

Cross-Domain Security Transfer: Techniques from other security domains may transfer to ML security: software supply chain security practices could improve model registry security, network security monitoring could inform ML API monitoring, and traditional incident response frameworks could adapt to ML-specific incidents. Research exploring these transfers could accelerate ML security maturity.

9 Conclusion

As organizations increasingly deploy deep learning models in production environments, security has emerged as a critical concern spanning technical, operational, and organizational dimensions. This survey provides a comprehensive analysis of security practices for ML deployment infrastructure, examining access control, runtime security, infrastructure protection, and operational best practices.

9.1 Summary of Contributions

This survey makes four key contributions to ML deployment security:

First Infrastructure-Centric Survey: Unlike previous surveys that examined ML attacks and defenses at the model level [27], this work focuses specifically on production infrastructure security—containers, orchestration platforms, serving frameworks, and deployment pipelines where security failures most commonly occur [3, 6]. Synthesizing 68 papers across security and systems communities (Sections 3-6), the analysis offers infrastructure practitioners comprehensive guidance previously unavailable from scattered sources.

Integrated Defense-in-Depth Framework: Section 7 demonstrates how access control, runtime security, infrastructure protection, and operational practices must integrate for effective defense. The taxonomy organizes 45+ security mechanisms while identifying critical dependencies—infrastructure isolation enabling access control, runtime monitoring informing authorization decisions. These dependencies reveal why isolated security measures provide insufficient protection for production ML systems.

Systematic Gap Analysis: Section 8 reveals five critical disconnects between research and practice. Configuration complexity causes security failures despite available features. Performance overhead (15-30%) renders adversarial defenses impractical for production use. ML-specific security tooling remains immature. Organizational silos separate security expertise from ML knowledge. Emerging threats from LLMs and edge deployments outpace defensive research. These gaps explain why research advances so often fail to reach production environments.

Actionable Practitioner Guidance: Rather than generic security advice, the survey provides specific, contextual guidance through comparative analysis tables (Tables 1-3), quantified performance overhead data (Table 3), and prioritized recommendations. It answers practical questions: when should teams use RBAC versus ABAC (Table 2)? Which authentication mechanisms suit different deployment contexts (Table 1)? How can organizations balance performance-security tradeoffs (Section 8.1.2)? This practical orientation directly addresses the needs of infrastructure teams deploying ML systems.

9.2 Key Takeaways for Practitioners

Organizations deploying ML systems should prioritize five key areas:

(1) **Defense-in-Depth:** No single security mechanism suffices. Effective protection requires multiple complementary controls: access control preventing unauthorized access, runtime monitoring detecting abuse, infrastructure isolation limiting blast radius, and operational practices sustaining security over time.
(2) **Secure Defaults:** Retrofitting security after deployment rarely succeeds. Configure systems with secure defaults appropriate for ML workloads from the outset—disable unnecessary features, enable network policies, enforce non-root containers, and implement least-privilege access controls during initial deployment rather than adding them later as an afterthought.
(3) **Continuous Monitoring:** Authentication attempts, API access patterns, model predictions, and infrastructure behavior all require comprehensive monitoring. ML-based anomaly detection helps scale this monitoring to the high volumes typical of production ML workloads.
(4) **Cross-Functional Collaboration:** Effective ML security demands teams that bridge security, ML, and operations expertise. This requires investment in cross-functional training, establishing shared responsibility for security outcomes, and creating processes that enable collaboration between teams with very different backgrounds and priorities.
(5) **Pragmatic Security:** Perfect security is neither achievable nor necessary. Balance security requirements against performance constraints and operational complexity. Focus on mechanisms that provide substantial risk reduction with acceptable overhead, while deferring more exotic techniques until they mature and prove their value.

9.3 Research Directions

Based on gaps identified throughout this survey, we identify critical research directions organized by their technical scope and potential impact on ML deployment security:

Critical Gaps and Immediate Needs. These directions address fundamental limitations in current practice where progress would have substantial practical impact:

(1) **Efficient Adversarial Defenses:** Current adversarial defense techniques impose 15-30% performance overhead, severely limiting their adoption in production environments. Research should target defense mechanisms with ¡5% overhead suitable for real-world deployment. Promising approaches include lightweight detection methods, efficient preprocessing transforms, and hardware-accelerated defense implementations that leverage existing GPU capabilities.
(2) **Automated Security Configuration:** Misconfiguration remains the primary cause of security failures in ML deployments. Tools for automated security policy generation, configuration validation, and vulnerability detection could dramatically reduce these risks. Such tools must handle ML infrastructure specifics—Kubernetes NetworkPolicies, RBAC configurations, container security policies—while remaining practical for teams without deep security expertise.
(3) **Production-Ready Anomaly Detection:** Bridge the gap between research prototypes and production systems by developing anomaly detection systems that handle concept drift, minimize false positives (¡1%), and operate with ¡10% overhead. Evaluate these systems in real deployment contexts with representative attack scenarios.

Important Research Directions. These directions address significant challenges requiring methodological advances and sustained research effort:

(1) **Privacy-Utility Optimization:** Explore techniques providing stronger privacy guarantees with less accuracy degradation. Research should quantify privacy-utility tradeoffs across different model architectures, deployment contexts, and privacy requirements [8].
(2) **ML Security Benchmarks:** Establish shared datasets, benchmark suites, and evaluation frameworks grounded in production scenarios. Include realistic attack patterns, representative workloads, and standardized metrics enabling comparison across security techniques.
(3) **Cross-Platform Security Standards:** Develop security standards spanning multiple ML frameworks, serving platforms, and deployment environments. Standardization would reduce configuration complexity and enable portable security policies across heterogeneous infrastructure.

Foundational Challenges. These directions require fundamental advances in theory, methodology, and technology:

(1) **Formal Verification for ML Security:** Adapt formal verification techniques to ML security properties, providing mathematical guarantees about security policies, isolation mechanisms, and defense effectiveness.
(2) **Hardware Security Integration:** Investigate security implications and opportunities from emerging hardware: confidential computing enclaves for model protection, specialized AI accelerators with security features, and quantum-resistant inference systems.
(3) **Emerging ML Paradigm Security:** Examine security challenges for large language models, multimodal systems, federated learning, and edge AI deployments. These paradigms introduce fundamentally new attack surfaces requiring novel defense approaches.

9.4 Closing Remarks

Securing ML deployment infrastructure requires technical mechanisms, operational practices, and organizational culture working in concert. While significant challenges remain—particularly in bridging research advances with production constraints—the path forward is clear: defense-in-depth architectures combining multiple security layers, automation reducing manual security burden, cross-functional collaboration bridging expertise gaps, and continued research addressing gaps identified in this survey.

As ML systems become increasingly critical to business operations, healthcare, finance, and public safety, security cannot remain an afterthought. Organizations must treat ML security as a first-class concern, investing in appropriate mechanisms, building necessary expertise, and maintaining security postures through continuous monitoring and improvement. The techniques, tradeoffs, and guidance presented in this survey provide a foundation for building secure, reliable, and compliant ML systems for production use.

References

[1] Outchakoucht, A., Es-Samaali, H., and Leroy, J.P. (2017). *Dynamic Access Control Policy based on Blockchain and Machine Learning for the Internet of Things*. International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 8, No. 7, 2017.

[2] Nobi, M.N., Gupta, M., Praharaj, L., Abdelsalam, M., Krishnan, R., and Sandhu, R. (2022). *Machine Learning in Access Control: A Taxonomy and Survey*. arXiv preprint arXiv:2207.01739, July 2022.

[3] Sultan, S., Ahmad, I., and Dimitriou, T. (2019). *Container Security: Issues, Challenges, and the Road Ahead*. IEEE Access, Vol. 7, pp. 52976-52996, 2019. DOI: 10.1109/ACCESS.2019.2911732

[4] Karn, R.R., Kudva, P., Huang, H., Suneja, S., and Elfadel, I.M. (2023). *Cryptomining Detection in Container Clouds Using System Calls and Explainable Machine Learning*. IEEE Transactions on Services Computing, 2023.

[5] Zhang, X., Zhao, P., Jaskolka, J., Li, H., and Lu, R. (2024). *SecMLOps: A Comprehensive Framework for Integrating Security Throughout the Machine Learning Operations Lifecycle*. arXiv preprint, 2024.

[6] Eken, B., Pallewatta, S., Tran, N.K., Tosun, A., and Babar, M.A. (2024). *A Multivocal Review of MLOps Practices, Challenges and Open Issues*. ACM Computing Surveys, 2024. DOI: 10.1145/3747346

[7] Ahmad, T., Adnan, M., Rafi, S., Akbar, M.A., and Anwar, A. (2024). *MLOps-Enabled Security Strategies for Next-Generation Operational Technologies*. ACM Conference Proceedings, 2024. DOI: 10.1145/3661167.3661283

[8] Murakonda, S.K. and Shokri, R. (2020). *ML Privacy Meter: Aiding Regulatory Compliance by Quantifying the Privacy Risks of Machine Learning*. USENIX Security Symposium, 2020.

[9] Al Rahat, T., Long, M., and Tian, Y. (2022). *Is Your Policy Compliant? A Deep Learning-based Empirical Study of Privacy Policies' Compliance with GDPR*. ACM Conference on Computer and Communications Security (CCS), November 2022. DOI: 10.1145/3559613.3563195

[10] Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., Rajashekhar, V., Ramesh, S., and Soyke, J. (2017). *TensorFlow-Serving: Flexible, High-Performance ML Serving*. Workshop on ML Systems at NIPS, 2017.

[11] Beck, N., Stein, B.J., Helmer, L., and Wegener, D. (2024). *Evaluation of Tools and Frameworks for Machine Learning Model Serving*. Fraunhofer Institute for Intelligent Analysis and Information Systems, 2024.

[12] Qiu, H., Mao, W., Patke, A., Cui, S., Jha, S., Wang, C., Franke, H., Kalbarczyk, Z., Başar, T., and Iyer, R.K. (2024). *Power-aware Deep Learning Model Serving with μ-Serve*. USENIX Annual Technical Conference (ATC), July 2024.

[13] Newman, H.B., Legrand, I.C., Galvez, P., Voicu, R., and Cirstoiu, C. (2003). *MonALISA: A Distributed Monitoring Service Architecture*. International Conference on Computing in High Energy and Nuclear Physics (CHEP), March 2003.

[14] Nokovic, B., Djosic, N., and Li, W.O. (2024). *API Security Risk Assessment Based on Dynamic ML Models*. Royal Bank of Canada Technical Report / McMaster University.

[15] Priya Bansal. (2022). *Study on Integration of FastAPI and Machine Learning for Continuous Authentication of Behavioral Biometrics*. International Journal of Engineering Research and Technology (IJERET).

[16] Hu, V.C. (2021). *Machine Learning for Access Control Policy Verification*. NIST Internal Report 8360, National Institute of Standards and Technology, September 2021.

[17] Soni, K. and Kumar, S. (2019). *Comparison of RBAC and ABAC Security Models for Private Cloud*. International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (Com-IT-Con), Feb 2019.

[18] Rao, U.P., Sahani, G.J., and Patel, D.R. (2010). *Machine Learning Proposed Approach for Detecting Database Intrusions in RBAC-Enabled Databases*. Second International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2010.

[19] Pasham, S.D. (2021). *Graph-Based Models for Multi-Tenant Security in Cloud Computing*. International Journal of Modern Computing (IJMC), Volume 2021.

[20] Jasti, A., Shah, P., Nagaraj, R., and Pendse, R. (2010). *Security in Multi-Tenancy Cloud*. IEEE International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2010.

[21] Djosic, N., Nokovic, B., and Sharieh, S. (2024). *Machine Learning in Action: Securing IAM API by Risk Authentication Decision Engine*. Royal Bank of Canada, Technology & Operations.

[22] Boxwala, A.A., Kim, J., Grillo, J.M., and Ohno-Machado, L. (2011). *Using Statistical and Machine Learning to Help Institutions Detect Suspicious Access to Electronic Health Records*. Journal of the American Medical Informatics Association (JAMIA), Vol. 18, pp. 498-505.

[23] Nobi, M.N., Krishnan, R., Huang, Y., Shakarami, M., and Sandhu, R.S. (2022). *Toward Deep Learning Based Access Control*. ACM Conference on Computer and Communications Security (CCS), April 2022. DOI: 10.1145/3508398.3511497

[24] Bou Nassif, A., Abu Talib, M., Nasir, Q., and Dakalbab, F.M. (2021). *Machine Learning for Anomaly Detection: A Systematic Review*. IEEE Access, Vol. 9, pp. 78658-78700, June 2021. DOI: 10.1109/ACCESS.2021.3083060

[25] Wang, S., Balarezo, J.F., Kandeepan, S., Al-Hourani, A., Chavez, K.G., and Rubinstein, B. (2021). *Machine Learning in Network Anomaly Detection: A Survey*. IEEE Access, Vol. 9, pp. 152379-152396, November 2021. DOI: 10.1109/ACCESS.2021.3126834

[26] Zhao, S., Chandrashekar, M., Lee, Y., and Medhi, D. (2024). *Real-Time Network Anomaly Detection System Using Machine Learning*. University of Missouri–Kansas City Technical Report.

[27] Papernot, N., McDaniel, P., Sinha, A., and Wellman, M.P. (2018). *SoK: Security and Privacy in Machine Learning*. IEEE European Symposium on Security and Privacy (EuroS&P), 2018.

[28] Katiyar, N., Tripathi, S., Kumar, P., Verma, S., Sahu, A.K., and Saxena, S. (2024). *AI and Cyber-Security: Enhancing Threat Detection and Response with Machine Learning*. Educational Administration: Theory and Practice, Vol. 30(4), pp. 6273-6282, 2024.

[29] Detection at Run-time. (2018). *Machine Learning For Security: The Case of Side-Channel Attack Detection at Run-time*. Security Conference.

[30] Murphree, J. (2016). *Machine Learning Anomaly Detection in Large Systems*. DRS Technologies Technical Report, 2016.

[31] Gholami, A. and Srivastava, A.K. (2020). *Comparative Analysis of ML Techniques for Data-Driven Anomaly Detection, Classification and Localization in Distribution System*. IEEE Transactions on Smart Grid, 2020.

[32] Siriwardena, P. (2020). *Advanced API Security: OAuth 2.0 and Beyond*. Apress, Second Edition, 2020. DOI: 10.1007/978-1-4842-5880-6 (Performance analysis in Chapter 4: Token validation overhead 2-5%)

[33] Naylor, D., Finamore, A., Leontiadis, I., Grunenberger, Y., Mellia, M., Munafò, M., Papagiannaki, K., and Steenkiste, P. (2014). *The Cost of the "S" in HTTPS*. ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT), 2014. DOI: 10.1145/2674005.2674991

[34] Li, W., Lemieux, Y., Gao, J., Zhao, Z., and Han, Y. (2019). *Service Mesh: Challenges, State of the Art, and Future Research Opportunities*. IEEE International Conference on Service-Oriented System Engineering (SOSE), 2019. DOI: 10.1109/SOSE.2019.00026 (Performance overhead analysis: mTLS in service mesh adds 8-15% latency)

[35] A SURVEY. (2023). *AI-Driven Container Security Approaches for 5G and Beyond*. 5G Security Conference.

[36] AKRAM AL-HOURANI. (2021). *A Combined Rule-Based and Machine Learning Approach for Automated GDPR Compliance Checking*. International Conference for Artificial Intelligence and Law (ICAIL), June 2021. DOI: 10.1145/3462757.3466081

[37] USER. (2024). *An AI Framework to Support Decisions on GDPR Compliance*. INTREPID Project - Italian Public Administration.

[38] Machine Learning Models. (2020). *Data Minimization for GDPR Compliance in Machine Learning Models*. arXiv preprint arXiv:2008.04113, August 2020.

[39] USER. (2024). *Data Compliance Complexities and Preventive Architecture Principles*. GRDJEV Journal, Vol. 9, Issue 100022.