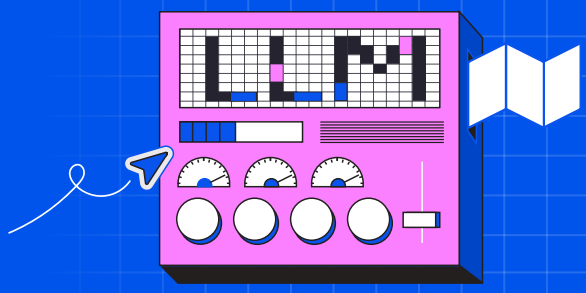
LLM Security Best Practices

In 2025, over 750 million applications are expected to use large language models (LLMs). That's not just scale—that's a seismic shift. But as adoption accelerates, security hasn't kept pace. LLMs bring entirely new classes of risk—from prompt injection attacks to model manipulation and data leakage—that standard controls just aren't designed to handle.



Whether you're a CISO overseeing AI deployments, a security engineer hardening infrastructure, or part of the GRC team shaping policy, you need a plan that works now—not after your chatbot leaks PII or your LLM API becomes a new attack vector.

This checklist offers practical, implementation-ready steps to guide you in securing LLMs across their lifecycle, mapped to real-world threats.

The evolving LLM security landscape

LLMs are fundamentally different from other software, even traditional ML solutions. They're not deterministic, don't follow fixed logic trees, and are trained on massive, often opaque datasets. That means traditional guardrails (think static code analysis or signature-based threat detection) won't catch threats like:

Reconnaissance	Resource Development	Initial Access	AI Model Access	Execution
6 techniques	12 techniques	6 techniques	4 techniques	4 techniques
Search Open Technical Databases	Acquire Public AI Artifacts	AI Supply Chain Compromise	AI Model Inference API Access	User
Search Open AI Vulnerability Analysis	Obtain Capabilities	Valid Account	AI-Enabled Product or Service	Command and Scripting Interpreter
Search Victim-Owner Website	Develop Capabilities	Evade AI Model	Physical Environment Access	LLM Prompt Injection
Search Application Repositories	Publish Poisoned Datasets	Exploit Public-Facing Application	Full AI Model Access	LLM Plugin Compromise
Active Scanning	Develop Capabilities	Evade AI Model	Physical Environment Access	LLM Prompt Injection

Click to see full Mitre Atlas Matrix

- Output leakage of sensitive context from previous sessions
- Model poisoning via corrupted fine-tuning datasets
- Unmonitored API usage leading to data exfiltration

But these risks aren't hypothetical. They're already happening across the very platforms your teams are experimenting with. For example, ChatGPT can be coerced into generating advanced malware if it thinks it's role-playing, and Gemini is powering cyberattacks for hacking groups.

As organizations rush to capitalize on the promise of generative AI, security is often playing catch-up as LLMs roll out in production—typically before risk teams even weigh in.

Your essential LLM security checklist

LLMs are now embedded in everything from customer support and software development to risk analysis. As your organization deepens its use of LLMs, this checklist offers a practical, actionable framework to help you secure deployments without losing momentum.

Across five core domains—data input/output, model and code, infrastructure, governance, and user access—we'll break down what threats you're defending against, why they matter, and what steps you can take right now to reduce risk without slowing down innovation.

Let's get into the action.

1 Securing LLM data inputs & outputs

Goal: To prevent unauthorized data access, leakage, or manipulation through LLM interactions

LLMs process user-provided prompts and return dynamic, often unpredictable outputs. That flexibility is what makes them powerful, but it's also where attackers strike first. Input/output security is your front line of defense.

1.1 Prevent prompt injection attacks

What prompt injections are: Prompt injection involves techniques that manipulate or alter the intended instruction flow of an LLM. Attackers may insert adversarial text to override built-in safeguards or to trick the model into revealing sensitive content. This kind of attack leverages the model's natural language processing to bypass control boundaries.

Why it matters: Attackers can exploit LLM logic to bypass restrictions and extract sensitive data such as PII, IP, and other confidential assets.

Action items:

- Implement **prompt validation and instruction filtering** to identify and block adversarial prompts.
- Use **allowlist/denylist patterns** for known-safe prompt components (e.g., How do I reset my password?) vs. known-dangerous prompt components (e.g., Ignore previous instructions) via regular expressions or NLP-based pattern matching to enforce boundaries around what users can input.
- Run regular **red teaming** focused on adversarial prompts to test your defenses against real-world manipulation attempts.

1.2 Protect against data leakage

What data leakage is: Accidental or malicious exposure of sensitive input/output data across LLM sessions. It is important to note that leakage may stem both from contextual carryover between sessions and from latent patterns in training data that inadvertently expose sensitive details.

Why it matters: LLMs can inadvertently "remember" or echo back information from prior inputs or training data, possibly leading to privacy violations.

Action items:

- Mask or tokenize **sensitive inputs** before processing to prevent PII, credentials, or proprietary content from leaking.
- Implement strict **context windowing and TTL** for prompts to ensure prior interactions don't influence new outputs.
- Use **context-aware data loss prevention (DLP) systems** to detect policy violations in real time and stop data exfiltration.

2 Securing models & code

Goal: To protect the integrity of models, training pipelines, and underlying code

Your LLM is only as trustworthy as its code and the data it was trained on. This section covers how to secure models from tampering, poisoning, or corruption—whether in training, deployment, or updates.

2.1 Verify model integrity

What model integrity is: Ensuring the model you deploy hasn't been tampered with or replaced

Why it matters: Malicious actors can introduce backdoors or sabotage model logic without detection to produce intentionally biased or malicious output.

Action items:

- Use **hash verification** for model binaries and weights to detect tampering or unauthorized changes.
- Implement **signed model checkpoints** and audit logs to confirm source and version provenance.
- Use **SBOMs (software bills of materials)** for every model version, deployment, and update event to ensure full traceability of model components.

2.2 Secure the training/fine-tuning pipeline

What training security is: Locking down the systems and data used to train or fine-tune your models

Why it matters: Poisoned training data or compromised code can stealthily alter model behavior.

Action items:

- Audit **training data** sources and contributors before training or fine-tuning to avoid introducing vulnerabilities.
- Segment **dev/test/prod pipelines** to reduce the attack surface and isolate compromise.
- Use **CI/CD and source control** to track provenance and prevent unauthorized changes.

- **Scan training datasets** for exposed sensitive data (e.g., credentials, PII) to prevent leakage and reduce poisoning risk.
- **Adopt reproducible builds** and containerized training environments that allow you to re-run experiments and compare outputs to detect any unintended modifications.

2.3 Prevent model poisoning

What model poisoning is: Attackers inject data designed to degrade model performance or introduce malicious behaviors.

Why it matters: Subtle poisoning is hard to detect but can cause security, reputational, or compliance fallout. For LLMs, the risk is even more significant because they are often trained on massive datasets without complete data quality assurances.

Action items:

- Scan datasets for **adversarial samples or backdoor triggers** to catch vulnerabilities at the source.
- Apply **differential privacy** (via TensorFlow Privacy, PyTorch Opacus, or OpenDP) where applicable to reduce the risk of memorization or sensitive data exposure by adding noise to training data.
- Incorporate model **explainability tools** to identify and audit unexpected model outputs or logic paths (outliers).

3 Securing infrastructure & APIs

Goal: To safeguard the foundation supporting your LLM deployments

Even the most secure model is vulnerable if the infrastructure around it is exposed. Infrastructure and API security are about locking down the pipes, servers, and endpoints that power your AI stack. Misconfigured or over-permissive cloud environments are the prime target for attackers.

3.1 Harden hosting environments

What environment security is: Ensuring your LLM compute and storage environments aren't exposed or misconfigured

Why it matters: Your LLM can be perfectly trained and still be compromised via a public S3 bucket.

Action items:

- Isolate **LLM workloads** using container or VM sandboxing to prevent lateral movement and sandbox execution.
- Implement **zero-trust networking** and **least-privilege access policies** across your deployment to ensure only authorized users can access LLM infrastructure.
- Regularly patch **hosting environments** and rotate **credentials** to avoid zero-day exploitation and credential theft.

3.2 Secure LLM APIs

What LLM API security is: Controlling how applications and users interact with your models programmatically

Why it matters: LLM APIs can leak data, be abused for enumeration, or become DDoS targets.

Action items:

- Use **API gateways** with strict rate limits and auth tokens to prevent abuse and credential stuffing.
- Implement logging and anomaly detection for **API usage** to detect abnormal behavior, like unexpected volume spikes or unauthorized access.
- Encrypt **data** in transit and at rest, even within VPCs, to ensure safe and compliant traffic.

3.3 Mitigate supply chain risks

What supply chain security is: Managing third-party risks from pre-trained models, libraries, or hosted services

Why it matters: AI supply chains are opaque and can introduce vulnerabilities deep in your stack.

Action items:

- **Vet third-party models** via security attestation before integration to ensure they meet your baseline security and compliance standards.
- Use registries with **tamper-evident signatures** for model downloads and container images to verify model authenticity and detect corruption.
- Maintain **SBOMs** and dependency graphing for all model dependencies to gain visibility into upstream risks.

4 AI governance, compliance, & monitoring

Goal: To gain control, visibility, and resilience around LLM usage

You can't secure what you don't monitor—and you can't govern what you haven't defined. This domain focuses on policies, monitoring, and response capabilities that help organizations scale LLM use responsibly. Visibility is the first step to security.

4.1 Introduce AI security policies

What policy enforcement is: Defining clear rules and expectations around acceptable use of LLMs

Why it matters: Shadow AI usage and model sprawl can expose organizations to unmanaged risk.

Action items:

- Establish **AI visibility agents** to detect and track new AI assets in your environment before deployment.
- Publish an **AI acceptable use policy (AUP)** to clarify safe and unsafe LLM behaviors.
- Require **approval workflows** for LLM-related projects and model deployments to ensure proper review and oversight of AI initiatives.

4.2 Implement continuous monitoring

What continuous monitoring does: Provides real-time visibility into model usage, behavior, and drift

Why it matters: Malicious activity or hallucinated responses stay unnoticed without monitoring.

Action items:

- Set up **logging** for prompts, outputs, and user metadata to support auditing, debugging, and compliance needs.
- Use **behavior baselines** to detect anomalies in model interaction patterns.
- Apply **AI-aware SIEM integrations** for correlation across threat intelligence feeds.

4.3 Conduct regular threat modeling

What threat modeling does: Identifies threats specific to your LLM architecture and use cases

Why it matters: LLMs introduce novel threat surfaces not covered by traditional AppSec frameworks, and threat modeling helps prioritize where controls are most needed.

Action items:

- Involve **red/blue teams** in LLM deployments to simulate real-world attacks.
- Use MITRE ATLAS or OWASP **top threats** for AI to guide threat model development.
- Update **threat models** regularly to stay relevant as models, data, and use cases evolve.

5 Securing user access & identity

Goal: To control who can access LLMs and what they can do

LLMs aren't just technical assets—they're tools that people interact with. Misuse by insiders or external users is one of the fastest paths to data loss or reputational damage. Strong identity controls are non-negotiable.

5.1 Enforce role-based access control (RBAC)

What RBAC access control does: Restricts access to LLM models, endpoints, and capabilities based on roles

Why it matters: Not every user or app should have full LLM access.

Action items:

- Use **IAM policies** based on job function and use case to control prompt and model access.
- Separate dev/test/staging/production access to reduce the blast radius and enforce clear boundaries.
- Limit **high-privilege operations** (e.g., retraining) to admins to prevent accidental or malicious misuse.

5.2 Monitor and audit AI interactions

What access security does: Tracks who accessed what, when, and why for your LLMs

Why it matters: Forensic visibility supports detection, compliance, and post-incident review.

Action items:

- Log all **user/prompt** interactions with metadata (timestamp, location, role) to enable forensic analysis and compliance reporting.
- Flag **interactions** containing sensitive terms or potentially dangerous instructions to detect misuse or leakage attempts in real time.
- Feed **audit logs** into your SOC for correlation with other infrastructure activity and alerting.

Future-proof your AI security program

Securing LLMs isn't a one-off project; it's an ongoing effort. It requires a comprehensive approach that encompasses data integrity, model robustness, infrastructure protection, governance, and user access controls. And as models evolve, so will the threats.

To get ahead of that curve, consider implementing a phased roadmap based on maturity levels—starting with high-impact controls (like prompt filtering and API hardening), then expanding into AI-specific governance, red teaming, and policy automation.

The key? Embed **security by design** into every phase of your LLM lifecycle—from data pipelines to model deployment to user interaction. That's where AI security posture management (AI-SPM) comes in.

Operationalize your LLM security with AI security posture management (AI-SPM)

AI security posture management (AI-SPM) is a structured, lifecycle-aware framework. It identifies, prioritizes, and mitigates AI-specific and LLM-specific risk end-to-end—from data ingestion to model behavior, infrastructure, compliance, and beyond. With AI-SPM, you can:

- ☑ Proactively detect misconfigurations, model vulnerabilities, data leakage, and model misuse sensitive data specific to your organization
- ☑ Monitor AI activity in real time across APIs, users, and outputs
- ☑ Enforce governance and access controls based on model sensitivity and risk
- ☑ Align AI usage with regulatory and compliance expectations

You've already got the checklist—now it's time to put it into action.

Ready to fast-track your LLM security posture? See how Wiz's AI-SPM can help you build a secure-by-default AI environment, and explore what an AI security assessment report looks like in practice!