Prompt Injection Defense Strategies

Advanced techniques for securing LLM applications

15 min read | January 2026 | Security

KEY TAKEAWAYS

- Injection is not a bug—it is an emergent property of probabilistic systems
- Never trust the model to enforce security
- The propose-and-validate pattern removes authority while preserving usefulness
- Prompt injection is solved by better architecture, not better prompts

Advanced Techniques for Securing LLM Applications Beyond Sanitisation

Abstract

Prompt injection is widely discussed and poorly understood. Most mitigation strategies proposed today focus on surface-level input filtering, pattern matching, or instruction rewriting. These approaches create a false sense of security while leaving the underlying system architecture fundamentally exposed.

This paper takes a deliberately uncompromising position. Prompt injection is not an input validation problem. It is a trust boundary failure.

Large language models do not understand intent. They optimise token sequences. When systems implicitly trust prompts, context, or model outputs, injection is inevitable. Defending against it requires architectural controls, not clever string manipulation.

This paper outlines advanced, production-grade strategies to defend LLM applications against prompt injection by redesigning how trust, authority, and execution are managed.

1. Why Prompt Injection Is Inevitable

Prompt injection exists because LLMs do exactly what they are designed to do. They follow instructions probabilistically.

© 2026 Architetto. All rights reserved. | architetto.io

There is no reliable distinction between system instructions, developer prompts, retrieved context, and user input at the model level. All are tokens.

Any system that assumes the model will respect instruction hierarchy without enforcement is already compromised.

Injection is not a bug. It is an emergent property of probabilistic systems.

2. The Myth of Sanitisation

The most common defence strategy is sanitisation. Filter dangerous phrases. Escape characters. Rewrite prompts.

These techniques fail because attackers adapt faster than filters can keep up. Prompt space is effectively infinite.

Sanitisation treats symptoms. It does not address the underlying authority confusion.

3. Instruction Hierarchy Is Not Enforced

Most LLM frameworks rely on an implied hierarchy. System prompts override developer prompts. Developer prompts override user input.

This hierarchy is not enforced by the model. It is a convention.

Attackers exploit this by reframing instructions, using indirect requests, or embedding adversarial context in retrieved data.

Security assumptions based on politeness are not security assumptions.

4. Trust Boundaries Must Exist Outside the Model

The fundamental defence principle is simple.

Never trust the model to enforce security.

All trust boundaries must exist outside the LLM. Models generate suggestions. Systems decide whether those suggestions are allowed to execute.

This requires explicit separation between reasoning and action.

5. The Propose and Validate Pattern

© 2026 Architetto. All rights reserved. | architetto.io

One of the most effective architectural patterns is the propose-and-validate pattern.

The model proposes an action, decision, or response. A separate validation layer evaluates it against policy.

If validation fails, execution stops.

This pattern removes authority from the model while preserving its usefulness.

6. Capability Constrained Execution

Prompt injection becomes dangerous only when the model has implicit authority.

Remove that authority.

Systems should expose explicit capabilities that require validation. File access. API calls. External communication.

The model cannot invoke capabilities directly. It can only request them.

Requests are evaluated by policy engines, not language models.

7. Context Isolation and Scoping

Many injection attacks exploit overly broad context.

Retrieval pipelines often assemble large, heterogeneous context windows mixing trusted and untrusted data.

Defensive systems scope context narrowly. Only the data required for the task is included. Sensitive instructions are never placed in the same context as user input.

Context is a privilege, not a convenience.

8. Signed and Typed Instructions

Another advanced technique involves signing instructions.

System and policy instructions are cryptographically signed and injected into execution layers, not prompt text.

The model never sees the full authority context. It sees only what it needs to reason.

Typed instruction channels further reduce ambiguity. Reasoning input is separated from execution metadata.

© 2026 Architetto. All rights reserved. | architetto.io

9. Output Validation and Policy Enforcement

Even with strong input controls, outputs remain untrusted.

Every model output should be validated before use.

Validation includes schema checks, semantic constraints, policy evaluation, and risk scoring.

Outputs that violate constraints are discarded or downgraded.

Blindly executing model output is negligence.

10. Agentic Systems Multiply Risk

Agentic systems amplify prompt injection risk.

Injected instructions can propagate across agents, escalate privileges, and trigger external actions.

Defensive architectures enforce per-agent capability boundaries, intent verification, and execution quotas.

Agents do not trust each other. They trust the control plane.

11. Monitoring and Anomaly Detection

Prompt injection attempts often leave behavioural traces.

Sudden shifts in output structure, unexpected tool usage, or repeated policy rejections signal an attack.

Monitoring systems must observe not just inputs, but decision patterns.

Silence is not safety.

12. Why Most Defences Fail in Production

Most defences fail because they are retrofitted.

Security added after deployment must work around flawed assumptions.

Systems designed from the start with explicit trust boundaries are inherently more resilient.

Retrofitting control onto autonomous systems is always harder than constraining them upfront.

© 2026 Architetto. All rights reserved. | architetto.io

13. The Cost of Overconfidence

Many teams believe prompt injection is a theoretical risk.

It is not.

As LLMs gain access to sensitive data and operational tools, injection becomes a primary attack vector.

Incidents will not look like hacks. They will look like legitimate actions taken for the wrong reasons.

14. A Realistic Defensive Posture

No system can eliminate prompt injection entirely.

The goal is containment.

Reduce blast radius. Prevent escalation. Detect misuse quickly.

This requires humility. Models are powerful, not trustworthy.

15. Conclusion

Prompt injection is not solved by better prompts.

It is solved by better architecture.

Systems that treat LLMs as untrusted components, enforce capability boundaries, and validate every action can operate safely even in adversarial environments.

Anything else is wishful thinking disguised as engineering.

© 2026 Architetto. All rights reserved. | architetto.io