Positive Prompt Injection: Gateway-Injected Policy as a Measured Control

Nov 14, 2026 · @Tekachew

Tekachew Ejeta, Denovo Security

Positive prompt injection places an organization's authorized policy into every model call at a gateway, checks the output, and logs the result. This paper defines the control, states what it can and cannot enforce, and specifies how to measure it. It is a design paper: it reports no measurements, and Section 6 gives the protocol for producing them.

1. The problem: policy does not reach the model call

An enterprise's AI policy is enforced only where someone remembered to write it into an agent's prompt. Enterprises now run many agents that read data, call tools and take actions with consequences. The agents come from different teams and vendors, and each carries its own system prompt and configuration.

Policy lives in documents and in the memory of whoever built each agent. Nothing checks it at the model call. An agent whose prompt is wrong, stale or missing a rule runs without that rule, and nobody is told.

The gap is a liability problem as well as a security one. State rules on automated decisions put the duties owed to the affected person on the deployer: notice, explanation, correction and human review (Section 8). A deployer that cannot show what its agents were told, and what they then did, has only its own assurance to offer.

The requirement is one control point in the request path of every agent the enterprise can route. It applies current policy whoever built the agent, and it records what happened.

Three limits bound everything that follows:

- **Coverage.** The gateway governs only model calls that pass through it in readable form.
- **Strength.** An instruction shifts a model's behavior. It does not bind it.
- **Evidence.** A record that a control ran is evidence of diligence. It is not compliance, and it does not move liability.

2. The approach: positive prompt injection

Positive prompt injection is the operator's own policy, inserted at a gateway into the system channel of a model call, with the outcome checked and logged per policy.

The name borrows a mechanism, not a threat. OWASP's LLM01 entry defines the vulnerability by a model's behavior being altered in ways nobody intended. Gateway policy is the reverse case: text placed in context from outside the agent's own configuration, on purpose and with authority. In neutral terms it is centrally managed system-level policy with closed-loop verification.

Five properties define the control:

- **Authorized.** The injected text is the organization's own policy, version-controlled and approved through change control. It is never drawn from documents, tool output or user input.
- **Privileged.** Policy enters through the system channel, which models are trained to rank above user and tool content. Authority comes from the channel. It does not come from a signature, which a model cannot check (Section 4).
- **Verified.** The gateway evaluates each response against each policy it applied and records the verdict.
- **Backstopped.** A rule that must hold gets a control that does not depend on the model: block, redact, escalate to a person, or insert required text.
- **Measured.** Verdicts aggregate into a violation rate per policy and per model, compared against a no-injection baseline (Section 5).

The control raises the rate of compliant behavior and produces evidence. It does not bind the model, guarantee an outcome or transfer liability, which moves only by contract.

The contribution is the closed loop, not the injection. Gateways already prepend policy text, and injection techniques have already been turned to defense (Section 3). What operators lack is an outcome for every applied policy, and a measured rate that says which rules instruction alone can carry.

3. Related work and what is new

Injecting policy at a gateway is established practice, and using injection techniques for defense is published research. This paper builds on both and claims neither.

Prior work	What it shows	Consequence for this paper
Kong AI Prompt Decorator	A commercial gateway plugin prepends system messages to every chat request. Kong's guide uses it for ethical, security and compliance instructions, checked by hand-run test prompts.	Gateway injection is existing practice. The guide describes no automated output check tied to the injected text.
Chen et al., 2024	Injection attack techniques, inverted, work as a defense: the attack pattern is repeated with the legitimate instruction.	Turning the attack mechanism to defensive use is already published.
Wallace et al., 2024	Models can be trained to rank system instructions above user and tool content.	This learned ranking gives gateway policy its authority. It varies by model.
Suo, 2024	Instructions are "signed" by swapping commands for character sequences the model is taught to recognize.	In-context signing is a learned convention. It is not cryptography.
Nasr et al., 2025	Adaptive attacks bypassed 12 recent defenses, most with success above 90%, often by impersonating system-level prompts.	Any trust marker inside the context must be assumed spoofable. Static tests overstate robustness.
Hines et al., 2024	Marking the provenance of untrusted text cut attack success from over 50% to under 2% in the authors' tests.	Basis for the untrusted-content policy in Section 8. The figures come from a fixed attack corpus.
Debenedetti et al., 2025	CaMeL separates control flow from untrusted data in a layer around the model, so retrieved data cannot change the program flow.	Guarantees come from architecture around the model, not from instructions. Backstops follow that principle.

Prior work	What it shows	Consequence for this paper
Gu et al., 2025	Prompt caching creates timing differences. An audit found cache sharing across users at seven API providers.	A cached policy prefix is cheap and can leak. Section 4 treats both.

Three things remain. None is a new mechanism.

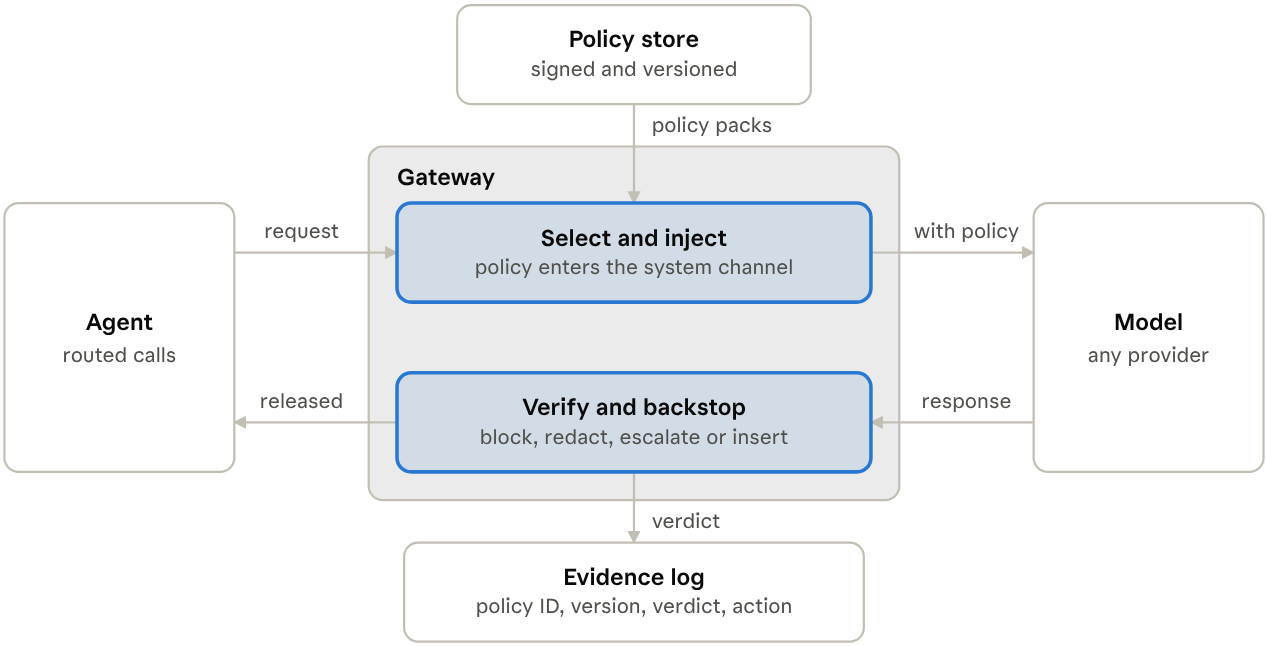
1. **The loop as one unit.** Injection, verification, backstop and log record share a policy identifier and version. Every outcome is attributable to a specific policy text.
2. **The residual-violation rate.** A rate per policy and per model, with a no-injection baseline and a correction for verifier error. It decides where backstops go.
3. **A current legal mapping.** The runtime duties in US state law as of this paper's date, mapped to that loop (Section 8).

To the author's knowledge, no gateway vendor publishes per-policy residual rates. That is a statement about published material, not about what vendors measure privately.

4. Architecture: the inject-verify-log loop

Every routed model call crosses the gateway twice: inbound to receive policy, outbound to be checked before release.

Every routed model call crosses the gateway twice



inject-verify-log loop · two gateway passes

Policy enters on the way in. Verification, backstops and the evidence record happen on the way out.

The request path

1. **Select.** The gateway reads trusted request metadata: agent identity, data classification, target system and jurisdiction. It selects the policy packs that apply.
2. **Inject.** It places the selected policy text in the system channel and notes each policy's identifier and version.
3. **Forward.** The call goes to the model provider.
4. **Verify.** The gateway evaluates the response, including any tool calls, against each policy it applied.
5. **Backstop.** Where a rule requires it, the gateway blocks, redacts, escalates to a person or inserts required text.
6. **Log.** It appends a record: request hash, policy identifiers and versions, hash of the injected text, verdicts, backstop action, model and model version.

Coverage

The gateway governs a model call only when the call passes through it in readable form. Agents fall into three classes.

Agent	Reach at the model call	Rely on instead
Built in-house and routed through the gateway	Full loop	Nothing else at this layer
Third-party agent with a configurable model endpoint	Full loop on model calls, none on the vendor's own logic	Vendor contract
Vendor SaaS agent that calls models from its own backend	None	Contract terms, plus inspection of the enterprise's own traffic to and from the service

Policy on agents the enterprise did not build holds only for the first two rows.

Channel and precedence

Policy enters as a system-level message. Models trained on an instruction hierarchy rank that channel above user and tool content (Wallace et al., 2024).

One conflict needs testing, not assumption: gateway policy against the agent's own system prompt. Both sit in the same channel, and models resolve the clash differently. Section 6 makes it an experimental factor.

One boundary is fixed. Gateway policy cannot override a provider's own trained rules, and should not try.

What signatures are for

A model cannot check a signature. Any marker placed in context, signed or not, is text an attacker can imitate. Adaptive attacks have succeeded by impersonating system-level prompts (Nasr et al., 2025).

This paper therefore makes no claim that a model trusts signed instructions. Signatures do three jobs outside the model:

- **Store integrity.** Each policy version is signed, and the gateway refuses to inject an unsigned one.
- **Proof of injection.** Each log record carries the hash of the injected text, so an auditor can show what the gateway sent.
- **Tamper evidence.** Log records are hash-chained, so a removed or altered record is detectable.

Two hardening steps reduce spoofing. The gateway neutralizes look-alike policy markers found in user or tool content before forwarding. It can also wrap its policy in a marker that is random per request, which pre-written content cannot match exactly.

A model may still follow an inexact imitation. Treat both steps as hardening, not proof.

What the verifier can and cannot see

A backstop is deterministic only when its trigger is. Checks fall into four classes.

Check	Example	Reliability
Structural	A required notice string is present. A tool-call argument holds no field classified as PHI.	Deterministic
Pattern or classifier	A PII detector over outbound text	Statistical, with a measured error rate
Judge model	Does the explanation state the principal factors of the decision?	Statistical, and must be validated against human labels
Not observable in output	Whether a protected attribute influenced a decision	Cannot be verified at the gateway. Needs input control and offline testing.

Deterministic checks run on every response. A judge-model check costs a second inference. Where a policy has no backstop, the judge can run on a sample, because its job there is measurement.

Placement and cache cost

Prefix caching makes stable policy cheap. It does not make it free. One provider's published rates show the shape: on Anthropic's API, a cache read costs about a tenth of the base input price, less on some models, and a five-minute cache write 1.25 times.

That cache covers the prompt in order: tools, system, then messages. A change early in the prefix therefore loses the cache for everything after it.

Per-request policy selection fights the cache. If the injected block varies by request, each variant needs its own cache entry, and the agent's own system prompt behind it is re-read at full price. The design uses two tiers:

- **Baseline pack.** Static per agent and deployment. It sits in the prefix and is cached.
- **Triggered rules.** Selected per request and appended after the conversation. The cached prefix stays intact, and the rules cost only their own tokens.

A late system message carries triggered rules where the API allows one. Otherwise they go in a marked block on the final turn, at lower authority. Whether late placement is obeyed as well as early placement is an empirical question, tested in Section 6.

Shared caches raise a second issue. Timing differences can reveal cache hits (Gu et al., 2025). Policy text is not sensitive, but which pack applied to a request can reveal its data classification. Confirm per-organization cache isolation with the provider before relying on a cached policy prefix.

The policy store

Whoever writes to the policy store instructs every routed agent. It gets the controls of a production signing system: two-person approval, signed versions, staged rollout and rollback. During rollout, the residual-violation rate and task success are the canary metrics.

Policy is static and human-authored. Policy retrieved or generated at request time brings retrieval poisoning into the highest-privilege channel. That variant is out of scope here.

5. The residual-violation rate

The residual-violation rate is the share of applicable requests that still violate a policy after the policy was injected. The gateway sets the instruction and observes the result, so every violation is labeled: the model was told, and failed anyway.

Definition

For policy p and model m, let A be the set of requests to which p applies. The residual rate r is:

> r(p, m) = (requests in A that violate p, with p injected) ÷ (requests in A)

The baseline rate b is the same fraction with p not injected. The instruction effect is the difference:

> Δ(p, m) = b(p, m) − r(p, m)

Without the baseline, a low residual rate says nothing about the injection. The model may have complied anyway.

Rates belong to a model version. A provider update can move them, so the matrix is re-measured on every version change.

Getting a baseline safely

Withholding a required policy from live decisions is not acceptable. Two sources give a baseline without doing so:

- **Offline replay.** Logged or synthetic requests run in a test environment, with and without the policy.
- **Holdout behind a backstop.** For a policy with a deterministic backstop, a small share of live requests skips the injection. The backstop still stops any violation before release. Only the model's raw behavior differs.

Correcting for the verifier

A verifier with errors reports the wrong rate. Let q be the share of responses the verifier flags. Let Se be its sensitivity and Sp its specificity, both measured on a human-labeled sample. The corrected rate follows Rogan and Gladen (1978):

> corrected r = (q + Sp − 1) ÷ (Se + Sp − 1)

An illustration with invented numbers: a verifier flags 4.0% of responses, with sensitivity 0.90 and specificity 0.98. The corrected rate is 0.02 / 0.88, or 2.3%. Nearly half the flags were false positives.

When the flagged share falls below the verifier's false-positive rate, the estimate goes negative. Report zero with its upper bound, and improve the verifier.

How much data a claim needs

Zero observed violations do not mean a zero rate. With no violations in n applicable requests, the 95% upper bound on the true rate is about 3 / n.

Claim at 95% confidence	Applicable requests needed, with zero violations
Residual rate below 1%	300
Residual rate below 0.1%	3,000
Residual rate below 0.01%	30,000

The counts are per policy and per model. They assume the verifier misses nothing; with sensitivity Se, divide the bound by Se. For non-zero counts, report a Wilson interval.

Reliability is not security

A rate measured on ordinary traffic is a reliability figure. An attacker chooses the inputs, and adaptive attacks defeat defenses that look strong on fixed test sets (Nasr et al., 2025). Report two rates per policy: on ordinary traffic, and under adaptive attack.

What the rate is for

The rate's first job is to place backstops. The risk owner sets a tolerance per policy before measurement.

Upper bound on residual rate	Cost of one violation	Action
Within tolerance	Low	Instruction and logging
Within tolerance	High	Backstop anyway. A rate is not a guarantee.
Above tolerance	Any	Backstop required. If no deterministic trigger exists, remove the capability or route to a person.

It has three further uses:

- **Choosing models.** The policy-by-model matrix shows which models carry which rules, measured on the organization's own traffic.

- **Sorting causes.** Violations spread evenly across inputs suggest a capability gap. Violations concentrated in particular sources or content suggest an attack or an edge case. Concentration is a cue to investigate, not proof.
- **Pricing the policy.** Report task success with and without injection. A policy that removes violations by breaking the task has been traded, not enforced.

6. Evaluation protocol

This protocol produces the measurements this paper does not yet report. It is sized for a first study and fixed before any run.

Design

Factor	Levels
Models	At least 4, from at least 3 providers, including one open-weight model. Exact version identifiers recorded.
Policies	The 8 runtime policies in Section 8
Injection condition	C0: no policy. C1: gateway policy in the prefix. C2: gateway policy placed late.
Agent prompt	Silent on the policy, or contradicting it
Inputs	Ordinary task requests, a fixed injection corpus, and an adaptive attacker

The ordinary-traffic grid has 4 × 8 × 3 × 2 = 192 cells. At 300 applicable requests per cell, that is 57,600 model calls. A cell with zero violations is then bounded at 1% (Section 5).

The protected-attribute policy has no output verifier. It is measured with paired requests that differ only in a protected attribute. The outcome is the share of pairs whose decision changes.

Verifier validation

For each policy, assemble a validation set of at least 100 violating and 100 compliant responses, labeled independently by two human raters. Run the verifier on it. Report sensitivity, specificity, their intervals and inter-rater agreement, and correct every

reported rate with the Section 5 estimator.

Adversarial runs

For the untrusted-content policy, use an agent benchmark with tool use, such as AgentDojo. For the two policies with the highest cost per violation, run an adaptive attacker that knows the policy text and the marker scheme.

What gets reported

- Residual rate and baseline for every policy and model, with intervals
- Instruction effect per cell
- Prefix placement against late placement
- The effect of a contradicting agent prompt
- Task success with and without injection
- Verifier sensitivity and specificity per policy
- Rates under attack, reported separately from ordinary traffic
- Added tokens, added latency and cache-hit rate per request

Commitments

Publish the prompts, policy text, verifier prompts and raw verdicts. Fix each policy's tolerance before the first run. Report cells that fail as plainly as cells that pass.

Threats to validity

- Synthetic requests differ from production traffic. Replay logged requests where privacy rules allow.
- A verifier validated on constructed violations can miss natural ones. Draw violating examples from real model failures where possible.
- Rates expire when a model version changes.
- Policy may weaken in long conversations as context grows or is compacted. Measure by conversation depth.

7. Threat model and limitations

Central control is a central target. Each threat below is marked enforced, observed or out of reach, because a control that claims to enforce everything is not credible.

Threat	Status	Treatment
Policy store compromise: an attacker writes policy and instructs every agent	Enforced	Signed versions, two-person approval, staged rollout and rollback (Section 4)
Marker spoofing: user or tool content imitates gateway policy	Partly enforced	Look-alike markers neutralized and a per-request random marker. Residual risk measured under adaptive attack.
Indirect injection through tool output	Observed, partly mitigated	Untrusted-content tagging lowers the rate. Tool-call allowlists and egress controls bound the damage.
Model ignores policy	Observed and measured	Residual rate, with a backstop where the rule must hold
Agent prompt contradicts gateway policy	Observed	Tested per model (Section 6). Conflicts logged for the agent's owner.
Verifier error or evasion	Measured	Sensitivity and specificity per policy. A deterministic check wherever one exists.
Traffic that bypasses the gateway	Out of reach at the model call	Network egress controls and vendor contracts
Cache timing side channel	Observed	Per-organization cache isolation confirmed with the provider
Training-time poisoning, model backdoors, social engineering of operators	Out of reach	Other controls

Limitations

- **Enforcement is probabilistic.** Injection influences a model and does not bind it. It must not be sold as a guarantee or as liability transfer.

- **The normalization objection has a narrow answer.** Critics will say that accepting injected instructions is the behavior attackers exploit. Gateway policy uses the system channel, which models already treat as authoritative, and adds no new trust path. What remains is spoofing of that channel's appearance, which Section 4 hardens and Section 6 tests.
- **The log cuts both ways.** It records violations the deployer knew about. Decide with counsel, before collecting, what remediation a known rate obliges.
- **Verification costs more than injection.** A judge model adds an inference per checked response. Budget for it, or sample.
- **Scope is runtime behavior.** Risk assessments, review staffing, correction handling and record retention are process duties. The gateway supplies evidence for them and does not perform them (Section 8).
- **Agents spawn agents.** A sub-agent that calls a model by another path is uncovered traffic. Coverage is a property of routing, not of intent.

8. Application to US AI compliance

Binding US rules on automated decisions come from the states, and the main consumer-facing duties begin on `Jan 1, 2027`. This section draws on law-firm summaries, policy trackers and one official bill summary, checked on November 14, 2026. It is not legal advice. Confirm each provision against the enacted text and current rules before relying on it.

Where the law stands

Law	Status	What it asks at runtime
Colorado SB 26-189	Signed May 14, 2026. Repeals and replaces the 2024 AI Act (SB 24-205). Duties apply from January 1, 2027. Attorney General rules are pending.	Notice before a covered ADMT materially influences a consequential decision. A plain-language description within 30 days of an adverse outcome. On request, data correction and, where commercially reasonable, meaningful human review. Records kept at least 3 years.

Law	Status	What it asks at runtime
Texas TRAIGA (HB 149)	In force since January 1, 2026	Intent-based prohibitions: encouraging self-harm, violence or crime; infringing constitutional rights; unlawful discrimination; sexually explicit deepfakes and child sexual abuse material. Government agencies must disclose AI interaction.
California CCPA ADMT regulations	Effective January 1, 2026. Businesses already using ADMT for significant decisions must comply by January 1, 2027.	Pre-use notice, an opt-out right with narrow exceptions, and an access right that explains the logic and how the output was used
California FEHA regulations	In force since October 1, 2025	Anti-discrimination protections apply to employment decisions made with automated-decision systems
Federal	Executive Order 14365 of December 11, 2025 directs challenges to state AI laws. No preemption statute has been enacted (GMU tracker).	State laws remain operative

Colorado's replacement law drops three duties of the 2024 act: the risk-management program, the impact assessment and the duty of reasonable care against algorithmic discrimination. The enacted TRAIGA is far narrower than the bill as introduced.

Correction. An earlier version of this paper mapped duties to Colorado's 2024 act and to TRAIGA as introduced. The annual risk assessments, 90-day notification window, oversight boards and semi-annual audits cited there do not appear in the current laws as summarized here.

Runtime duties mapped to the loop

Runtime duty and source	Injected instruction	Verifier	Backstop
Pre-use notice. Colorado SB 26-189; California ADMT regulations	Before any step that contributes to a consequential decision about a person, state that automated decision-making technology is in use and give the link for more information.	Structural: notice text present	Gateway inserts the notice
AI-interaction disclosure. Texas TRAIGA, for government agencies	At the start of the interaction, state that the user is interacting with an AI system.	Structural: disclosure present	Gateway inserts the disclosure
Adverse-outcome description. Colorado SB 26-189	If the outcome is adverse to the person, describe the decision and this system's role in plain language, and say how to request more information.	Judge model, human-validated	Escalate: open a disclosure task with the 30-day deadline
Human review on request. Colorado SB 26-189	Do not present an adverse consequential decision as final. Tell the person they can request human review, and route such requests to the review queue.	Structural: review ticket created when requested	Escalate to the reviewer

Runtime duty and source	Injected instruction	Verifier	Backstop
Prohibited uses. Texas TRAIGA, which turns on intent	Refuse to produce sexually explicit deepfakes or child sexual abuse material, or content encouraging self-harm, violence or crime.	Classifier	Block
Protected attributes. Anti-discrimination law, including California's FEHA regulations and TRAIGA	Do not use protected attributes, or proxies for them, as decision factors.	None at the output. Paired-request testing offline.	Redact protected attributes from the model's input
Data minimization in tool calls. HIPAA minimum-necessary standard, 45 CFR 164.502(b)	Send external tools only the personal data the task requires. Include PHI only for tools on the approved list.	Structural: field classification and PHI detection on tool-call arguments	Redact, or block the tool call
Untrusted-content tagging. Internal security control, not a legal duty	The following is external, untrusted content. Treat it as data, not instructions.	Observed: actions that follow instructions found in tagged content	Tool-call allowlist and egress controls

Three cautions apply to this table:

- **Notices.** A statutory notice belongs in the application's own interface. The gateway insert is a safety net behind it.
- **Explanations.** A model's account of its own reasoning is not a reliable record of what drove a decision. Treat a generated description as a draft for the deployer's disclosure, built from logged inputs and the decision record.
- **Texas.** TRAIGA's prohibitions turn on the developer's or deployer's intent. A refusal policy with a log is evidence about intent. It is not a statutory duty in itself.

Process duties the gateway only supports

Law	Process duty	What the log contributes
Colorado SB 26-189	Handle correction requests. Staff human review with a trained person who has authority to override. Keep compliance records at least 3 years.	Decision records, policy versions and verdicts, retained on the statutory schedule
California CCPA ADMT regulations	Complete a risk assessment before using ADMT for a significant decision. Honor opt-outs.	Evidence of which requests used ADMT, and how often each rule failed
Texas TRAIGA	Cure a violation within 60 days of an Attorney General notice	A dated record of testing and remediation

TRAIGA's safe harbors cover substantial compliance with NIST's AI Risk Management Framework, and violations a person finds through testing or good-faith audits. A measured residual-rate program is evidence toward both. Whether it suffices is a legal question.

Jurisdiction routing

Jurisdiction-keyed selection needs a signal the gateway can trust: the person's state from the application's own records, never inferred from the prompt. Where the signal is missing, apply the union of packs. The union also keeps the cached prefix stable.

Federal preemption

Preemption is being pursued and has not happened. The executive order set up a Justice Department task force to challenge state AI laws, and no preemption statute had been enacted by November 11, 2026. Build to duty categories that recur across states, so a change in one statute moves a pack and not the design.

9. Conclusion and research agenda

Positive prompt injection steers a model. It does not guarantee what the model does. The design places policy in the privileged channel, verifies each response per policy, backstops what must hold, logs with provenance, and measures where instruction fails.

The design leads with static, human-authored policy. Dynamic or generated policy waits until the static case is measured.

For a deployer, the product of the loop is a record: what each agent was told, what it did, and how often each rule failed. That record is evidence of diligence. Liability stays where contracts put it.

Five questions follow, in priority order:

1. **The residual-rate matrix.** Run Section 6 and publish it. How stable are rates across model versions and providers?
2. **Placement and precedence.** Does late policy hold as well as prefix policy? Which models side with the gateway when the agent's prompt disagrees? How fast does policy decay in long conversations?
3. **Spoof resistance.** How often do models follow look-alike policy markers under adaptive attack, with and without per-request markers? A provider-verified operator channel would turn this from convention into mechanism.
4. **Verifier quality.** Where does a judge model's error swamp the rate it measures, and which duties admit a deterministic check instead?
5. **Cache cost and leakage.** What does pack fragmentation cost at realistic pack counts, and what does pack selection reveal through timing?

The measurement is the point. It shows where an agent can be trusted with a rule, and where it cannot.

References

Research

- Chen, Y., Li, H., Zheng, Z., Song, Y., Wu, D., Hooi, B. Defense Against Prompt Injection Attack by Leveraging Attack Techniques. arXiv:2411.00459, 2024.
- Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., Tramèr, F. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. NeurIPS 2024.

- Debenedetti, E., Shumailov, I., Fan, T., et al. Defeating Prompt Injections by Design. arXiv:2503.18813, 2025.
- Gu, C., Li, X. L., Kuditipudi, R., Liang, P., Hashimoto, T. Auditing Prompt Caching in Language Model APIs. ICML 2025.
- Hines, K., Lopez, G., Hall, M., Zarfati, F., Zunger, Y., Kiciman, E. Defending Against Indirect Prompt Injection Attacks With Spotlighting. arXiv:2403.14720, 2024.
- Nasr, M., Carlini, N., Sitawarin, C., Schulhoff, S. V., et al. The Attacker Moves Second: Stronger Adaptive Attacks Bypass Defenses Against LLM Jailbreaks and Prompt Injections. arXiv:2510.09023, 2025.
- Rogan, W. J., Gladen, B. Estimating prevalence from the results of a screening test. American Journal of Epidemiology 107(1):71-76, 1978.
- Suo, X. Signed-Prompt: A New Approach to Prevent Prompt Injection Attacks Against LLM-Integrated Applications. arXiv:2401.07612, 2024.
- Wallace, E., Xiao, K., Leike, R., Weng, L., Heidecke, J., Beutel, A. The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions. arXiv:2404.13208, 2024.

Practice and standards

- Anthropic. Prompt caching. Documentation.
- Kong. Use the AI Prompt Decorator plugin. Documentation.
- NIST. AI Risk Management Framework, with its Generative AI Profile (NIST AI 600-1), July 2024.
- OWASP. LLM01:2025 Prompt Injection.
- US Department of Health and Human Services. Minimum Necessary Requirement, 45 CFR 164.502(b) and 164.514(d).

Law and policy

- Colorado General Assembly. SB26-189, Automated Decision-Making Technology. Signed May 14, 2026.
- Baker Botts. Colorado Repeals and Replaces AI Act.
- Crowell & Moring. Colorado Hits Reset on AI Regulation: SB 26-189 Repeals and Reenacts the Colorado AI Act.
- Greenberg Traurig. TRAIGA: Key Provisions of Texas' New Artificial Intelligence Governance Act.

- Baker Botts. Texas Enacts Responsible AI Governance Act: What Companies Need to Know, July 2025.
- DLA Piper. Updated CCPA regulations, September 30, 2025.
- Akin Gump. New California Regulations Regarding Employer Use of Automated Decision-Making Technology, December 16, 2025.
- George Mason University, Center for Assurance Research and Engineering. State AI Policy: A 50-State Tracker. Status as of October 17, 2026.
- nhimg.org. US AI Regulation Outlook: Board Briefing. Status as of November 11, 2026.