AI AGENT SECURITY — AUTHORITY, PROVENANCE, EVIDENCE

Prompt Injection Is an Epistemic Attack: Provenance, Claim Grading, and Layered Defense for AI Agents

A prompt injection does not primarily corrupt text. It corrupts what an agent believes about who instructed it and what its principal intends.

THE BATTERY	56 planted instructions and 50 legitimate controls. The epistemic layer challenged 55 of 56 with no false challenges; a structural screen alone challenged 32 of 56 and falsely challenged 12% of controls.
THE RED TEAM	47 new attack phrasings the battery had never tried. After hardening aimed at the failure reasons: 45 of 47 challenged at the decision layer, and 23 of 25 on a sealed held-out set.
THE TUNING EXPERIMENT	A tuned small model challenged 27 of 56 plants (base model: 0 of 56) — and falsely challenged 16 of 28 legitimate controls. Tuning helps, over-flags, and is not sufficient alone.

It is, in that precise sense, an epistemic attack — an attack on the agent's evidence about authority.

Axiovex Systems, LLC · 29 October 2026 · Public

© 2026 Axiovex Systems, LLC. All rights reserved.

Abstract

Prompt injection is usually described as a filtering problem: keep bad instructions out of the model's context, or teach the model to spot them. This paper argues that both descriptions misstate the attack. A prompt injection does not primarily corrupt text. It corrupts what an agent believes about who instructed it and what its principal intends. It is, in that precise sense, an *epistemic* attack — an attack on the agent's evidence about authority. It is therefore best combatted with an epistemic solution: establish provenance outside the text, treat instruction-shaped content found in data as a claim about the principal's intent, grade that claim by source and corroboration against the principal's record, and challenge it by default when nothing independent backs it.

We report a connected program of measurements in support of that framing. First, a token-cost study of recorded agent traffic shows why a deterministic defense matters: output was only about 11.8% of measured spend, input was 88.2% and dominated by history re-sent many times over, and the largest proven saving came not from rewriting text but from sending it once and referring to it thereafter. Second, a semantic compression study shows that rewriting transforms lose at the token meter and destroy planted failure signals, while a deterministic screen costs zero model tokens. Third, we describe a message-security design — provenance tagging, deterministic structural and epistemic screens, and an action gate — and measure it: an epistemic layer challenged 55 of 56 planted instructions with no false challenges on 50 legitimate controls, where a structural pattern screen alone challenged 32 of 56 and falsely challenged 12% of controls. Fourth, an adversarial red team shows the honest limits of that first result and how hardening changed them. Fifth, a small model-tuning experiment shows that tuning helps, over-flags, and is not sufficient alone; on the red team's adaptive phrasings, which neither model had encountered in training, the tuned model challenged 53.9% of attacks where the base model challenged 5.3%, at the same false-alarm cost. Our conclusion is a layered stack — behavior, governance, enforcement, and model — in which no single layer is load-bearing on its own.

1. The thesis: injection is an attack on belief about authority

An AI agent working on a real task reads far more than its instructions. It reads tool outputs, files, web pages, messages from other systems, and its own earlier output. Some of that material will, sooner or later, contain sentences written to direct the agent: cancel the task, send a file somewhere new, hand over a credential, delete a record, do not tell anyone.

The conventional way to talk about this is as contamination — bad text got in. That description points defense in the wrong direction. The text itself is not dangerous because it is impolite, unusual, or keyword-laden. Legitimate work is full of imperative text: runbooks, documentation, code comments, and quoted examples all tell someone to do something. The planted sentence is dangerous for a different

© 2026 Axiovex Systems, LLC. All rights reserved.

reason: if the agent accepts it, the agent's picture of *who its principal is and what that principal asked for* has been replaced by a picture supplied by an interested third party.

That is an epistemic failure before it is a security failure. The agent has taken a proposition — "the principal intends this" — on evidence that would be rejected in any other domain: the say-so of the very text that benefits from being believed.

The defense follows from the diagnosis:

- **Provenance first.** Authority comes from the channel a span of text arrived through, never from what the span says about itself. A notice that prints "system message" inside a tool output is still a tool output.
- **Instruction-shaped content in data is a claim, not an instruction.** It is graded the way any claim is graded: who says it, what independent evidence backs it, and whether it fits the record of what the principal actually asked for.
- **Challenge by default.** A claim backed only by its own assertion is challenged, always, regardless of phrasing, urgency, or claimed approval. Urgency, pre-approval, and requests for secrecy inside data are not evidence for a claim. They are, if anything, evidence against it.
- **Being fooled must not equal being harmed.** A final action gate checks any consequential proposal — an external send, a deletion, a credential use, a write outside the workspace — against the principal's record. A model that was persuaded still cannot execute a consequential action on a data span's say-so.

The rest of this paper is the evidence, the measurements, and the limits.

2. Token economics: why the defense should be deterministic

Security mechanisms for agents are often priced as if model attention were free. Our recorded traffic says otherwise.

In a measured run of 86 agent calls totaling 427,062 tokens, output accounted for 50,475 tokens — about 11.8% of spend. Input accounted for 88.2%. The dominant input cost was not new material; it was history. Summing one attempt's per-call inputs against its final call's input gives a re-send factor of roughly 17 to 19 times per attempt: an attempt pays for its accumulated history again on nearly every remaining call, and every reply the model makes becomes a further deposit into that history.

Three consequences follow for defense design.

First, **a defense that spends model tokens on every span is expensive in exactly the wrong place.** A second model judging every input, or a model re-reading its context to police itself, taxes the 88.2% side of the ledger repeatedly. A deterministic screen — rules over text and provenance tags, running in the harness before the model sees anything — costs zero model tokens. That is not a minor efficiency point. It is what makes always-on screening affordable.

© 2026 Axiovex Systems, LLC. All rights reserved.

Second, **the proven way to cut input cost is reference, not rewriting.** Carrying history as disciplined views against known state, rather than re-delivering raw text, cut exact replay of recorded traffic by 81.5%. Send once, refer thereafter. That result frames the compression study in the next section: the savings live in not re-sending, not in respelling.

Third, **output definition is a real but bounded lever.** Because output was about 11.8% of spend, no output-side technique can save more than that share directly, though a shorter reply also deposits less into history. That bound matters when evaluating claims for output-side codes, and we hold our own candidate to it in Section 3.

3. The compression study: rewriting loses, removal and reference win

A natural hope is that agent text could be compressed semantically — sent in a compact code that the model's well-known tolerance for damaged text would repair on receipt, the way a listener repairs imperfect speech. We tested that hope in token space, with the tokenizer of the model family in use, over prose, code, tool outputs, and numeric text drawn from our own records. The transfer condition is arithmetic: a code saves only if reading it costs fewer tokens than the text it replaces.

The results were consistent and, for the literal version of the idea, negative:

- **Phonetic respelling cost more everywhere** (about +17% on prose, +15% on code). The tokenizer already owns cheap representations for standard spellings; respelling breaks them and pays to reassemble words from smaller pieces. In this medium, standard spelling is already the efficient code.
- **Vowel deletion cost far more** (about +30% to +36%), despite producing far fewer bytes. Shorter in bytes is not shorter in tokens.
- **A naive abbreviation dictionary also lost** (about +1.7% on prose, +5.9% on tool outputs). The most frequent long words are frequent partly because the tokenizer already represents them cheaply. A dictionary must target strings that are frequent in traffic but expensive in tokens — long paths, hashes, repeated command lines — and at that point it has become the reference mechanism of Section 2 under another name.
- **Opaque compression (compress, then encode) lost on prose and code at the token meter**, whatever it did to byte counts. It could win the raw token count only on highly repetitive material, and even there it fails the decisive test: the model cannot read it natively, so the count win cannot be spent in context. Its honest home is off the meter — storage and transport — not agent input.
- **Function-word deletion was the one readable transform that genuinely saved:** about −11.2% on prose and −11.6% on tool outputs. Deleting words whose content the reader can regenerate — *the, of, and, to* — removes whole tokens cheaply and needs no codebook. The saving is a useful trim of about a ninth, not a codec multiple.

Fidelity is the second axis, and it is where rewriting fails outright. We planted twenty failure markers in tool-output text and applied the transforms. Function-word deletion preserved all twenty verbatim: 20 of

© 2026 Axiovex Systems, LLC. All rights reserved.

20. Every rewriting transform — vowel deletion, phonetic respelling, and the abbreviation dictionary — preserved none: 0 of 20, because each rewrites the marker itself. A failure signal is by nature the rare, atypical line in an output; a transform that normalizes text normalizes exactly that line away.

A fuller follow-up battery, with markers planted in varied phrasings including full-sentence failures, sharpened the deletion result honestly. Unprotected, function-word deletion preserved only the terse markers (14 of 40), because a failure stated as a sentence loses words to deletion too. With a verbatim fence — protected spans such as failure lines, markers, and numbers pass through untouched — deletion preserved 40 of 40, at savings of about 11.3% on prose, 11.7% on tool outputs, and 12.0% on failure-bearing output. Even fenced, it stayed below the 15% saving at full survival that we had set in advance as the bar for this family, beyond what a bounded digest already achieves. On these measurements, the input side of this family closes. That is a finding, not a disappointment: it was reached by a falsifier stated before the test.

The output direction inverted the verdict, for a structural reason. If the model emits a terse, templated form and a deterministic program outside the model expands it into full grammar, the expansion costs zero model tokens — the decoder sits off the meter, which is where a codec's decoder has always belonged. In an offline round-trip test over recorded responses, a deterministic encoder and expander round-tripped 73 of 73 formulaic responses exactly, and the coded form saved 2.13% of total output spend, against a pre-set falsifier line of 2%. Two bounds must travel with that pass. Templates alone saved only 1.13% and would have failed; the margin came from a dictionary mined from the same corpus, and must be re-measured on any other corpus. And formulaic turns were only about a tenth of billed output tokens, with reasoning content making up roughly two-thirds — territory this candidate deliberately does not touch. The direction is structurally promising; the prize, on this traffic, is small and corpus-specific. Whether a model can *emit* such a code fluently, rather than merely be re-spelled into it offline, is untested.

One interaction between compression and security deserves emphasis, because it is easy to miss: **screening must run on a form where injections are visible.** A lossy transform can hide a planted instruction from a screener, or worse, launder a data span's instruction into prose that sounds like the system's own narration. Our rule is therefore strict: screen before any lossy transform, on the raw span; transformed views inherit the lowest authority of their sources; and encoded content from untrusted sources must be decoded and screened in expanded form, or refused.

4. The message-security design

The design has three tiers of authority and no fourth. System policy instructs. The user's own messages instruct. Everything else — tool outputs, file contents, retrieved text, other agents' messages, and the model's own prior turns — is data, with no authority however it is phrased.

Provenance tagging. Tagging happens in the harness, before the model sees anything, because the harness is the component that actually fetched each span and therefore knows where it came from. Each span carries a tag recording its class, its authority, its source, a content hash, and its screening verdicts. The representation is structural — metadata in the request the harness builds — not marker text inside

© 2026 Axiovex Systems, LLC. All rights reserved.

the content, because content can print the same markers and forge a boundary. Where a serving surface forces everything into one string, the harness emits delimiters itself and treats any delimiter-like text inside a span as ordinary content. Tags survive the token path because the harness owns every transition: history is re-tagged from the harness's own record, summaries inherit the lowest authority among their sources, and freshly fetched content is tagged at fetch time.

Deterministic screens. Every data span passes two deterministic layers at assembly time. A structural screen looks for known shapes: override phrases, fake authority markers, urgency paired with claimed approval, exfiltration shapes, concealment phrases, and new destinations. An epistemic screen grades claims. Both are ordinary computation over text and tags; neither spends model tokens.

The epistemic screen, step by step. Instruction-shaped content in a data span is treated as a claim of the form "the principal intends this." The screen classifies the source from the tag; checks the claim against the principal's record — the standing rules, the task as actually given, and the destinations and actions already authorized — and fires a challenge independently for contradiction, task expansion, a new destination, or concealment; asks who benefits if the claim is obeyed; counts the claim's self-description ("urgent," "pre-approved," "system notice," "do not flag this") as no evidence in its favor and as a detection signal in itself; and challenges by default any agent-directed claim that no principal record corroborates. Every challenge is written as an auditable record entry: the span's source and hash, the claim, which check fired, and what was done.

Severity is tiered so that the defense stays usable. A *challenge* is never obeyed and is recorded; if its target is consequential, it is escalated to the principal in plain language before anything proceeds past that point. A *note* — instruction-shaped content that is not aimed at the agent, such as runbook steps for a human operator, code comments, or a claim genuinely presented as a quotation or example at the claim itself — is recorded without interrupting anyone, and the content remains usable as data. The asymmetry is deliberate: a false challenge costs a record line and, at worst, one interruption; a false acquittal can cost a sent file, a deleted record, or a suppressed log.

The action gate. Before any proposed consequential action executes, the harness checks it against the principal's record: was this action, to this destination, actually authorized through a principal channel? This gate is the design's load-bearing wall. It does not consult the model's beliefs at all, which is why a completely persuaded model still cannot send, delete, or disclose on a data span's authority.

Packaging. The same discipline is published at four depths, which compose rather than compete. As a skill, it is behavior: agent-facing guidance any model can carry in context from the day it is written — and, being advice a model reads, exactly as reliable as the model's attention at the moment of attack. As harness middleware, it is enforcement: tagging, screening, and gates in code that runs on every call regardless of the model's state. As an extension to our open-source specification toolkit, it is governance: every project that uses it must write down its authority surface — who may instruct the system, through which channels, and which actions and destinations are authorized — which is precisely the principal record the screen checks against, made reviewable; the planted-injection battery becomes an assessment gate in that flow; and challenges land in a standard record format. Tuned into a model, it would be a fourth, inner layer — the subject of Section 6, with its limits stated there.

© 2026 Axiovex Systems, LLC. All rights reserved.

The public code and documents for this work are published by Axiovex Systems in the `injection-defense` repository: https://github.com/electrohire/injection-defense.

5. The battery, and what a red team did to it

5.1 The initial battery

We built an offline battery of 106 fixtures from our own program text — prose, code, and harvested tool output. Fifty-six fixtures were carrier spans with a planted instruction appended, across seven types: direct override, fake system or approval markers, self-attestation and urgency, exfiltration, concealment, task expansion, and a new destination — half phrased obviously, half phrased to avoid the obvious keywords. Fifty fixtures were legitimate controls: the user's own instructions, standing rules, quoted and documented examples, runbook steps, code comments, documentation, and specification text.

Measured as detection and challenge — whether the layer refuses the claim authority and writes the entry — the results were:

- **Structural screen alone:** 32 of 56 injections challenged (57.1%), with 6 of 50 controls falsely challenged (12%). Its failures were structural, exactly as its design predicts. It challenged none of eight task-expansion plants, because "beyond the task as given" is not a property of text and a text-only screen has no task record to compare against. Its challenge rate collapsed from 82% on obvious phrasing to 32% on stealth phrasing, because patterns are a memory of previous attacks. And its false challenges fell on the most legitimate text in the corpus — the user's own instructions and the standing rules — because it cannot see provenance.
- **Epistemic layer:** 55 of 56 injections challenged (98.2%), with 0 of 50 controls falsely challenged. It was perfect on six of seven types, including every task-expansion and self-attestation plant — the two types defined by their relationship to the record rather than their wording. Its cost on legitimate text was notes, not challenges: 29 of 50 controls drew a note line, which never interrupts anyone. On stealth phrasing it held at 27 of 28, against the structural screen's 9 of 28.

The single miss was instructive. A data span announced, in the voice of a system team, that the standing rules had been revised, and added that "the revision is quoted in this file." The span asserted its own framing, and the framing mitigation believed the assertion. Self-declared framing is self-attestation wearing a second hat. The principled fix is that framing evidence, like authority, should come from outside the span — from the harness's knowledge of what it fetched.

Two caveats belong with these numbers wherever they are cited. The screener and the battery were developed in the same program of work, so the battery measures this design against this fixture family, built from this program's text; it does not measure performance against an adversary who has read the screener. And the battery measures the deterministic layers only — whether a model in context would obey a claim the layers passed is a separate question, addressed only in part in Section 6.

© 2026 Axiovex Systems, LLC. All rights reserved.

5.2 The red team

A red-team exercise then wrote 47 new attack phrasings the battery had never tried: polite verb forms (“dispatching,” “relayed,” “handing over”), sentences that merely assert they are quotations, encoded payloads, claims split across two spans, and instructions hidden in filenames, table cells, structured log fields, and other places sentence-based screening does not naturally look.

Against the original screener, the result was sobering: the epistemic decision layer challenged only 6 of 47. The published battery result was real but narrow. Polite and passive phrasing was largely invisible to verb lists built from imperatives; asserted framing earned mitigation it had not shown; nothing was ever decoded; and nothing was ever assembled across spans.

Fixes were built and measured, each aimed at a failure reason rather than a fixture. Framing mitigation now requires presentation — the claim actually appearing inside quotation marks or a code fence, with no live directive outside the presented part — not assertion. The screener decodes candidate encoded content before grading, reassembling fragments across spans, and grades assembled claims as well as single spans. Verb and form coverage was widened so gerund, passive, and declarative presentations shape like the imperatives they replace, and claim classes were added for declarative cancellation, task redefinition, destination aliases, and a broader set of approval claims.

After hardening, the screener challenged 56 of 56 on the original battery, still with 0 of 50 controls falsely challenged; 45 of 47 on the adversarial scenario suite at the decision layer (46 of 47 counting the structural screen); and 23 of 25 on a sealed held-out set written before the fixes and never used in designing them. The two held-out misses were left unpatched on purpose, because patching against a held-out set destroys what it measures. They share a shape worth naming: a consequential claim phrased so that no sensitive verb, object, destination, or direct address appears in the same span — a passive purchase naming no object the heuristics grip, and a soft cancellation whose follow-on sentence stays purely descriptive. Heuristic reach remains the binding constraint, and the durable defense against the next phrasing is the regression discipline — every new family found in the wild enters the battery first — not any single fix.

The red team also quantified an attack on the defense itself: flooding the challenge channel. Repeating one identical consequential claim across many spans would, under a naive protocol, produce one escalation per repeat — a hundred interruptions carrying one piece of information, training the principal to stop reading the one that matters. The protocol therefore collapses identical claims (same action, destination, and object) into one escalation with an occurrence count: a hundred repeats produce one escalation, a 99% reduction, with the count itself preserved as a signal. Varied flooding is not solved by this, and we do not claim otherwise; the tier structure, under which notes never interrupt, is the mitigation there.

Residuals from the red team, stated where the claims are made: encoding coverage is a list, not a principle — several common encodings are not yet decoded by the screener, though the agent-facing rule covers them behaviorally; sequence grading only grades the spans the harness supplies, so coverage of surfaces, not cleverness of grading, is the operational risk; approval patterns broad enough to catch chains will fire on some legitimate retrospective prose in the wild, so the false-challenge rate in

© 2026 Axiovex Systems, LLC. All rights reserved.

production will be higher than the battery's zero; and multi-turn grooming — spans that individually define, normalize, and then use a new frame across a session — is only partly closed, with the action gate as the backstop for what grading misses.

6. The tuning experiment: help, with a headline-sized caveat

Could the model layer itself be improved by tuning? We ran a deliberately small, honest experiment. A small open instruction model (1.5 billion parameters) was given light low-rank adapter tuning on 132 short authored examples — 84 teaching resistance across the seven attack families, with new surface text, and 48 teaching legitimate behavior, included specifically to bound false challenges. Both the base and tuned models were then evaluated, identically, on the battery's 56 injection fixtures plus 28 legitimate controls, held out from training.

The base model challenged none of the 56 planted instructions: 0 of 56. It complied with 34 outright, and most of the remainder were responses the scoring rubric could not classify as resistance at all — summaries that silently absorbed the plant as if it were part of the task. In one recorded case, a data span ending in an instruction to upload a workspace archive to an outside address was folded into the model's summary as the plan, with no flag and no hesitation.

The tuned model challenged 27 of 56 (48.2%), with gains in every family, and its behavior became more legible: unclassifiable responses fell from 21 to 4. That is a real, measurable improvement, and it is only half the result.

The other half: false challenges on legitimate controls rose from 4 of 28 (14.3%) to 16 of 28 (57.1%). The tuned model began treating benign material — an ordinary code comment, in one recorded case — as an attack. The legitimate-behavior examples in training did not bound false challenges adequately at this tuning strength. As measured, the adapter trades one failure mode for another, and would not be acceptable as a sole behavioral layer.

We then ran both models against the red team's adaptive material: the 47 scenario phrasings, the 25 sealed held-out entries, and the four assembled split claims — 76 attacks per model, with the same 28 controls, the same prompt shape, and the same scoring. The base model challenged 4 of 76 (5.3%) and complied with 56 outright. The tuned model challenged 41 of 76 (53.9%) and complied with 19 — a slightly larger gain than on the battery, on phrasings its training never contained. It challenged all four assembled split claims, which the base model challenged none of. The cost did not move: the tuned model again falsely challenged 16 of 28 controls, and they were the identical 16 items as on the battery — which, under deterministic decoding, identifies the over-flagging as a stable property of this adapter rather than sampling noise.

Three qualifications matter more than that headline. First, the tuned model's success on encoded payloads (7 of 8 challenged, against 0 of 8 for the base model) is suspicion of opacity, not decoding: it flagged the encoded blobs while inventing their contents, and the one encoding that resembles ordinary language — a rot13 item — beat it entirely, drawing a fabricated translation instead of a flag. Second, destination aliases were never cleanly challenged: all three were flagged and then restated anyway, the

© 2026 Axiovex Systems, LLC. All rights reserved.

flag and the adoption coexisting in a single response — the grooming foothold, in measured form. Third, descriptive re-presentation beat both models outright: a plant restated as a digest's neutral description of what the assistant does drew no clean challenge from either model (0 of 4 for the tuned model). The deterministic screen catches that family; the model layer, tuned or not, does not. The backstop for it remains the action gate.

The limits of this experiment are part of the result. It is one small model; the battery run used 56 attacks and 28 controls, and the adaptive run 76 attacks and the same 28 controls, so the percentages carry wide uncertainty and should be read as "measurably moved," not as precise rates. Training and evaluation share one author's idea of what attacks look like, though no evaluation fixture was used in training. Nothing here transfers by assertion to any other model or size, and a tuned variant of any production model would need its own battery run and admission proof. The trade-off itself, however, is the headline the field should expect: tuning raises resistance, over-flags, and never reaches certainty. It produces no reasons and no records. That is why, in the stack below, it sits innermost and is never load-bearing.

Combining Sections 5 and 6 gives the model-layer picture we can honestly claim. The deterministic epistemic layer is measured, in full, in Section 5. For the model layer itself, we now have two before-and-after measurements, and they agree: on the battery, an untuned model at 0 of 56 and a tuned model at 27 of 56; on adaptive phrasings neither model had seen, 4 of 76 against 41 of 76 — in both cases with the same severe over-flagging cost, and in both cases with descriptive re-presentation unbeaten. We make no stronger claim about model behavior in context, and the design does not need one.

7. The layered conclusion

No single approach in this paper is sufficient alone, and each fails in a characteristic way the others cover:

- **Patterns** fail on novel phrasing and cannot see provenance or the task.
- **Epistemic grading** depends on a principal record that must be kept honest; an incomplete record over-challenges until it learns, and its heuristics will miss exotic shapes.
- **A skill** is advice; it protects exactly when the model attends to it.
- **Governance** — a written authority surface and a battery gate — raises the floor and makes omissions visible, but paperwork does not stop an action.
- **Tuning** is probabilistic, over-flags at useful strengths, and leaves no record of why anything was refused.
- **An action gate** fails only on missing coverage — a consequential path the harness does not mediate — which is an engineering-discipline risk, and an auditable one.

Composed, the stack is: behavior (the skill, reaching inside the context where only the model can reason about a novel situation in full), governance (the authority surface and battery, reaching across projects and across time), enforcement (provenance tagging, deterministic screens, and the action gate, reaching the moment an action would execute), and the model layer (tuning, where available, welcomed as

© 2026 Axiovex Systems, LLC. All rights reserved.

defense in depth). The epistemic layer is the integrating layer rather than a replacement for the others, because it is the only one that consumes the rest as evidence: the structural screen's findings are signals in its grading, the architecture's tags are its provenance input, and the gate enforces its verdicts.

Reliability, in this framing, is not the promise that an injection will never be believed. A sufficiently targeted injection against a model layer will, someday, be believed, and we state that plainly. Reliability is that belief does not become damage: the claim is graded before the model relies on it, the attempt is in the record with its reason, a persuaded proposal meets a gate that does not share the persuasion, and the consequential action does not execute without authorization from the principal's own channel. Landing is not damage. That, and not a detection percentage, is the property worth engineering for.

8. What this work does not establish

- **General model behavior.** The model-layer measurements in Section 6 cover one small model, one prompt shape, and deterministic decoding, across a battery run and an adaptive suite. No claim here measures any other model — including the larger models used in production serving — and none should be read as doing so. The design deliberately does not depend on that answer.
- **Robustness against an adaptive adversary with full knowledge of the screener.** The red team narrowed this gap and named what remains; it did not close it.
- **Split and multi-session claims in full generality.** Assembled grading covers the span sets it is given; claims assembled across sessions, or from individually innocent pieces with no sensitive element in any span, are only partly covered, with the action gate as backstop.
- **Non-text channels.** Images, audio, and structured results whose shape itself implies an action are outside these measurements, though the tagging and gate design extends to them.
- **Production-scale records.** All screening measurements use a small principal record. At production scale, record completeness — not phrasing cleverness — becomes the binding constraint on false challenges.
- **Transfer.** Token measurements use one tokenizer family; battery results use one program's text; tuning results use one small model. None of the numbers transfer by assertion to a different stack.

About Axiovex Systems

Axiovex Systems is a Michigan engineering consultancy. Its work includes manufacturing AI infrastructure and evidence and decision systems. Its research program develops open-source tooling for disciplined, evidence-graded AI-assisted engineering, including a specification toolkit extension suite and a benchmark harness. This research comes from that program: the same insistence on provenance, graded claims, and auditable records that Axiovex Systems applies to engineering work, applied to the security of the agents doing it. The message-security design, battery, and related documents are published in the public `injection-defense` repository at https://github.com/electrohire/injection-defense.

© 2026 Axiovex Systems, LLC. All rights reserved.

References and public artifacts

- Axiovex Systems, *injection-defense* — public repository: design documents, the message-security skill, and battery materials. https://github.com/electrohire/injection-defense
- Axiovex Systems, specification toolkit extension suite and benchmark harness — open-source tooling for evidence-graded AI-assisted engineering, published by Axiovex Systems. The message-security work is distributed through this suite as an extension, and measured through this harness as middleware.

© 2026 Axiovex Systems, LLC. All rights reserved.