SILENT MCP CONTRACT DRIFT, CAUGHT AT THE MOMENT OF USE

[mcpindex] — the trust-to-act layer for agent tool calls.

The tool your agent trusted on Monday can change on Tuesday, silently. mcpindex holds the call before your agent acts.

It pins every tool's contract and holds the call the moment that contract moves. A verdict is a change-tripwire, not a safety oracle: a HOLD means the contract moved, not that the call is dangerous.

Version 1.0 · Launch edition

`[ mcpindex ]` — the trust-to-act layer for agent tool calls.

⬡ The trust-to-act layer: it pins every tool's contract and holds the call the moment that contract moves.

> *The tool your agent trusted on Monday can change on Tuesday, silently. mcpindex holds the call before your agent acts.*

A passive sniffer can only alert. The gate can hold. That one-line contrast is why mcpindex is a layer you act *through*, not another scanner you read.

> *The gate earns the install. The network keeps it.*

A verdict is a **change-tripwire, not a safety oracle**: a HOLD means the contract moved, not that the call is dangerous. Route every HOLD to a human (§9 #1). "Holds before your agent acts" is precise: **in-session when the host re-lists, and from the persistent pin at next launch otherwise** — against an adversary who withholds `list_changed` the in-session catch degrades to that next-launch floor (§2, §10).

Version 1.0 · Launch edition

Abstract

mcpindex is the trust-to-act layer for agent tool calls: it pins every tool's contract and holds the call the moment that contract moves. Under the Model Context Protocol, the tool your agent trusted can change its contract between one call and the next — silently, with no version bump, from a server you do not run — and your agent acts on the change as if nothing moved. No host enforces a persistent contract baseline in-path today, and the protocol does not require one, so nothing in the standard publish-time stack catches that day-two silent change. That is the seam mcpindex closes, and the bet is bigger than the catch.

The day-one deliverable is the **local tier-0 gate**: zero-egress, standalone, complete on your host the day you install it, on the persistent stdio/SDK path — the value lands the first time a contract moves, not at install. The part that compounds, and the moat, is the layer the gate grows into: the cross-install verdict corpus, the outcome-flywheel calibration *it will accrue*, and the governance an ecosystem embeds against — recognized across every install rather than re-derived on each (§6). The gate is live today; the network accrues. The gate earns the install; the network keeps it.

⬡ Executive summary (read this if you read nothing else)

- **The wedge.** A free, open-source, local gate that pins each tool's contract and HOLDs the call the moment the contract moves — zero egress, on your host, the day you install it.

- **The runtime gap.** Every existing tool — registries, static scanners, the MCP spec's own review — checks a tool at publish; none enforce a persistent contract baseline in-path after your agent already trusts it (the one canonical statement of the gap; §1 prosecutes it). That is the only moment trust is actually spent, and the moment mcpindex owns.
- **The adversarial floor, stated with the headline.** Against an adversary who *withholds* the `list_changed` notification (a TOCTOU seam the gate cannot force a re-list to close today), the **in-session** catch is not guaranteed; the floor is **next-launch detection from the persistent pin**, and the in-session gap closes only when forced periodic re-list ships (§10). We do not claim a measured in-the-wild drift incidence rate — we argue from the protocol's structure and the disclosed attack classes [1]–[5] (§5.6).
- **Why now.** MCP adoption put attacker-supplied, un-versioned tool contracts inside every agent's context, and the 2025 security literature documented the rug-pull, tool-poisoning, and shadowing classes [1][2][3][5].
- **The moat.** A cross-install verdict corpus × an outcome-flywheel × a governance posture an ecosystem embeds against — copyable diff, uncopyable network (§6).
- **The ask.** Install the gate; design-partner the network.

> ⬡ **Reader-router. Product / buyer / press path (the ~7-page spine):** §1 problem → §3 the two surfaces → §4 architecture → §7 the proof → §6 the moat → §9 the limits → §11. **Security & procurement path:** §2 (the honest box) → §8 (deployment, trust boundary, procurement) → the Appendix (full adversary perimeter + questionnaire-grade math). Read the spine for the decision; read the security band for the audit.

○ Buyer fast-path — install, what it catches, what "quiet" means.

```
# one-click — installs the local gate, wires your hosts (with consent), holds drift on t
curl --proto '=https' --tlsv1.2 -fsSL https://mcpindex.ai/install.sh | sh
```

```
// TypeScript SDK — wrap a session your host already authenticated. An empty PreflightPi
// store TOFU-pins every tool on the first list_tools (no baseline to build).
import { wrap, PreflightPin } from '@mcp-index/sdk';   // npm i @mcp-index/sdk (scoped:
const wrapped = wrap(session, { pin: new PreflightPin(), serverId });
```

```
# Python today — the genuinely-shipping proxy/CLI path (peer to the TS SDK; §8.2):
uv tool install mcpindex-preflight        # the gate as a proxy/CLI — no host-config rew
mcpindex-preflight pin   <your-server>     # capture the baseline
mcpindex-preflight check <your-server>     # HOLDs the call if the contract moved
```

- **What it catches:** a tool that, after your agent already trusts it, silently *adds a required parameter*, *flips to destructive*, *removes a relied-on parameter*, *narrows a constraint*, *reduces an enum*, or *rewrites its description* — the changes that turn a trusted tool hostile before an irreversible call (§1, §4.3).
- **What "no banner" means:** *protected and quiet*. Day one the gate pins every tool you use and watches; the first banner fires the first time a contract moves, not at install. The first-10-minutes walkthrough (fire a HOLD yourself with `mcpindex-preflight`, read the pinned-inventory card) is §8.6.
- **Free, forever, for the gate:** unlimited local gating, all postures, every host, zero egress — free under the stated license. You pay only if you opt into the cross-install network or the enterprise tier (§6).
- The full install detail (flags, the `.mcpb` bundle, the locked-down-endpoint manual path, what `install.sh` rewrites) is §8.2. The ten honest limits are §9; you can hold drift before you read a word of them.

1. The problem: silent MCP contract drift

Your agent planned a call to `make_report` against the contract it read this morning. Between that plan and the call it now makes, the tool quietly gained a `required` parameter (`owner`), and the result it returns is about to be acted on irreversibly. No host enforces a persistent contract baseline in-path today, so the change reads as the normal state of the world, and the agent runs the call. Swap `make_report` for *delete a record, transfer a budget, escalate a permission* and the stakes are concrete. Under MCP this is not a corner case; it is structurally possible by design.

The runtime gap, in one line. Every existing tool (registries, static scanners, the MCP spec's own review) checks a tool *at publish*. None enforce a *persistent* contract baseline in-path *after your agent already trusts it*. That is the only moment trust is actually spent, and the moment mcpindex owns.

Where this sits in your stack — complement, not replace. mcpindex does not replace your registry or scanner; it fills the one window none of them watch. The table scores each tool on the only two axes that matter for runtime trust: *when* it checks, and whether it can *hold* a call or only *report* on it.

TOOL	WHEN IT CHECKS	CAN IT ACT, OR ONLY ALERT?
Tool registry / marketplace	at publish / listing	neither — it catalogs
Static scanner	at publish / CI	alerts (a finding, pre-deploy)
MCP spec security review	at design / publish	alerts (guidance, not enforcement)
Passive runtime sniffer	every call, observing	**alerts only** — it sees the call, cannot stop it
mcpindex gate	**every call, in-path**	**holds** — it stops the call before the agent acts

A registry tells you what a tool *was* at publish; a scanner flags a known-bad pattern before deploy; a passive sniffer can *see* the live call but, sitting out of the path, can only alert after the fact. **A passive sniffer can only alert; the gate can hold.** The contrast is scoped, not a superset claim: the gate holds on a contract *change* it can observe in-path against its pin; a sniffer alerts on live traffic it cannot stop. They watch different things — the gate is not "a sniffer plus enforcement." Keep your registry and scanner for the publish-time job they do well — mcpindex is the in-path layer that catches the day-two silent change they structurally cannot.

MCP standardized how an agent discovers and calls a tool. A server exposes a `tools/list`; each tool ships a *self-description*: a `name`, a `description`, an `inputSchema`, optional `annotations`, and an optional `outputSchema`. The description is not documentation the agent skims; it is instruction the model **conditions on**, exactly as it conditions on its system prompt. The self-description is the contract.

Three properties of that contract create the problem: it is **attacker-derivable** (supplied by a server you may not run; every field is an input you do not control); it **can change at any time, with no version bump** (a tool can quietly gain a `required` parameter, narrow an enum, flip an annotation to destructive, or rewrite its description between one call and the next); and **no in-path baseline is enforced** (the protocol does not require a host to pin contracts, and none does in-path today — a fresh `tools/list` returns whatever the server says *now*). The spec's `list_changed` notification exists precisely so a compliant host *can* track change (§2); nothing forces a persistent baseline, which is the gap mcpindex fills.

What the gate catches, in concrete buyer terms. In practice the gate catches the change that actually bites you: a tool that silently adds a *required* parameter, flips itself destructive, narrows a constraint, reduces an enum, removes a relied-on parameter, or rewrites its description after your agent already trusts it. That is the core. Everything in §2 and §9 is the honest perimeter *around* that core.

The mutation-after-trust attack. mcpindex defends a precise subclass: a contract that **moves after you already trusted it**, a runtime-timing refinement of the documented MCP rug-pull, not a new attack class. The 2025 MCP security literature establishes the shape: the *rug-pull*, where a server mutates a tool's contract after the user approved it [2]; *tool-poisoning / tool-description injection*, where the agent conditions on a hostile description as instruction [1]; *cross-tool shadowing* [3]; and the MCP specification's own *Security Considerations* on un-versioned, server-supplied tool definitions [5], an independent, non-vendor corroboration of the premise. The broader indirect-prompt-injection literature frames the root cause: an LLM cannot reliably separate trusted instructions from attacker-controlled data in its context [4]. mcpindex is the runtime detector for mutation-after-trust: it does not re-litigate whether a description was poisoned the first time you saw it (the screen's job, §3); it catches the moment a contract you already trusted *moves*.

The standard publish-time stack catches the lie at publish; nothing catches the silent change at runtime. mcpindex exists to close the runtime gap.

2. The threat model · *(For security & procurement review — the spine reader has the honest box in §4; go to §3.)*

> ⬡ **This begins the security-review band.** The narrative spine is §1 → §3 → §4 → §7 → §6 → §9 → §11; a product, buyer, or press reader gets the honest box around the win **in §4** (right after the mechanism is shown) and can skip straight to §3. This section, §2.5, and §8.1–§8.5 are the audit-grade perimeter for a security or procurement reviewer — read in full when you are doing the threat-model assessment, not the product evaluation.

The value of a detector is bounded by the threat model it actually defends. The gate's win is the reproducible HOLD in §7; this section draws the honest box around that win, so the win is credible. We state what the gate *does* catch, then the honest boundary around it.

The core the gate defends. Any **change** to a pinned tool's contract: a structural drift in the input schema, a destructive annotation flip, a description edit, an injection/exfil marker appearing in the input schema, output schema, or description, and a brand-new un-pinned tool appearing where the agent expects a known set. The list below is the honest perimeter around that core.

Adversary. The party that controls a tool's self-description: a malicious or compromised MCP server, a supply-chain takeover of a previously-honest server, or an insider who can push a contract change. The adversary controls every field and the timing of any change.

Adversary capabilities.

- Mutate any field of a tool's published contract at any time.
- Fire, or *withhold*, the `list_changed` notification.
- Embed instructions or exfiltration markers in any agent-consumed channel, including the `outputSchema`, which is unhashed and read by the agent but rarely inspected by humans.

The per-call check is against the last view the client re-listed. On every `call_tool` the gate diffs the tool view it currently holds against the pin; it refreshes that view (and re-pins) when the client re-lists. So "checks on every call" and "diffs the last re-listed view" are the same mechanism: a change is caught on the next call *after* the client re-observes the contract. **In-session detection therefore requires the host to re-list**, a precondition, not an assumption. A compliant host re-lists on a `list_changed` notification, and the shipping proxy re-pulls `tools/list` on that notification, so a server that fires `list_changed` is caught within the session. For a host that does *not* re-list within a session, the practical guarantee degrades to next-launch detection from the persistent pin (§4.2): a server that changes while the agent is offline is caught on the next launch.

Re-list behavior per supported host. The in-session guarantee's precondition is concrete per host: Cursor and Claude Desktop are verified to re-list on `list_changed`; Cline and Zed (the other two supported install hosts) are assumed spec-compliant pending per-host verification. For any host, the persistent-pin next-launch floor (§4.2) holds regardless.

Explicitly out of scope (the five highest-signal; the full perimeter is in the Appendix).

- **Drift that predates the pin (intrinsic).** The gate establishes its baseline trust-on-first-use (TOFU). A tool already compromised *before* you installed the gate is the baseline; a HOLD means the contract changed *versus your pin*, not that the pin was ever safe.
- **A malicious tool that never changes (intrinsic).** The gate is a *change* detector. A tool hostile from first sight that stays byte-identical produces no drift — the directory screen's domain (§9 #2).
- **Mutation with a withheld `list_changed`, against a client that re-lists only on notification.** This is the seam an adversary probes first. A server that mutates a tool **and withholds `list_changed`** against a client that *only* re-lists on notification leaves the gate diffing a stale view; the within-session drift is not seen until the next re-list. The single extractable residual-risk line: **against this adversary, in-session detection is NOT guaranteed today — the floor is next-launch detection from the persistent pin (the stdio/SDK path), and the in-session gap closes only when forced periodic re-list ships (§10).** The gate does not force a per-call `tools/list` round-trip (it is not free, and the MCP contract is that servers announce changes). The real mitigation is forced periodic re-list / startup re-validation, not corpus coverage (§10): the corpus keys on a hash the gate would only compute if it re-observed the contract, so more coverage cannot help a gate that has not re-listed.
- **Runtime behavior divergence with an unchanged contract.** A tool whose declared contract is stable but whose *behavior* quietly changes upstream is the behavioral tier's domain (§5), and even there the verdict clears or refutes (§9 #7).
- **Response-content injection.** A live attack class — tool-poisoning / prompt-injection delivered through a tool's *response content* [1][4] — lands in the data a tool returns at call time, on a byte-stable, pinned-clean contract, and §4.6's zero-body discipline means the response stream is deliberately never inspected. mcpindex does not defend it. This is not the weaker surface: it is the surface that *cannot be enforced in-path*. A response stream cannot be statically pre-cleared before the call, whereas a contract change can be caught deterministically before the agent acts. We defend the surface the gate can enforce and name the other (§9 #4).

Four further out-of-scope cases — **cross-server shadowing**, **list-vs-execution substitution**, **the composed gateway threat (withheld-`list_changed` × restart-re-TOFU)**, and **a compromised host** — are enumerated in full in the Appendix perimeter band, so this section reads as scope rather than a hedge parade. The out-of-scope set is the honest perimeter that makes the in-scope guarantee credible. *(The full adversary-perimeter enumeration of mcpindex's own surfaces — §2.5 — and the procurement quick-reference table also sit in the signposted "For security & procurement review" band, §8 and the Appendix.)*

2.5 Adversaries within mcpindex's own surfaces (in-scope, mitigated)

A gate in-path is itself an attack surface: an adversary can target *the gate*, not only the tool. **Five such surfaces are in scope and defended in code** — the tier-2 judge as escalate-only, the cloud tier-1 response as closed-vocabulary-validated and endpoint-pinned, the HOLD-banner field redaction, the held-and-empty corpus poison-resistance, and the adversarial HOLD-fatigue vector. The full per-surface enumeration, with its mitigations and the host-level availability / parser-DoS / pin-store-tamper model, is in the Appendix and §8.4 — pushed there so this section reads as scope, not a hedge parade.

3. The two surfaces over one question

mcpindex answers one question — *should my agent act on this tool call, right now?* — at two moments. The two surfaces sharpen it differently: the **gate** asks *"has the contract moved since I pinned it?"* (a change question, in-path); the **screen** asks *"was the published contract clean when I read it adversarially?"* (a prior, advisory). **Lead with the gate.** The directory screen is the prior; the gate is the wedge and the live check.

The drift gate (the live check, the wedge). The gate sits **in-path**. It observes the contracts your client already reads, pins them, and on every `call_tool` checks the live contract against the pin. On a silent change, the gate **HOLDs the call before the agent acts** — the moment trust is actually spent, when the agent is about to do something irreversible with a tool whose contract just moved under it. ("Before the agent acts" is precise about *when* the change is observed: **in-session when the host re-lists** the tool (the §2 precondition), and **at next launch from the persistent pin** otherwise — the body refines this lead in §2 and §9 #9; it does not walk it back.) A passive sniffer can only alert; the gate, the proxy, and the SDK wrapper are all in-path, so they can hold. The gate's *judgment* is a contract-diff, not a safety ruling; its *effect* is a real, enforced HOLD that stops the call so a human decides.

The directory screen (the prior). Before you wire a tool, the directory gives an advisory verdict on the *published* contract, read adversarially for hidden instructions, exfiltration patterns, prompt-injection payloads, and overclaims (a tool that says it "validates" a field it never checks). **One posture, three facets:** it is **advisory and semantic** (it reasons about a contract, it does not sit in your call path and never enforces — the in-path gate does), its conformance probe is **monitored, not enforced** (a failure surfaces in the verdict; it does not block a call), and its coverage is a **graduation milestone** that expands as the corpus grows, adversarial cases first. The verdict an agent reads is one of **four states**, the canonical screen vocabulary published on `/methodology` and `/trust`:

VERDICT	KIND	MEANING
ALLOW	decision	The published contract was screened and no finding crossed a deny threshold at the recorded evidence level; advisory, not a safety warranty; advisory until `expires_at`; the in-path gate, not the screen, enforces.
DENY	decision	The eval ran and a finding crossed the deny threshold (high-severity intent flag, conformance regression, poisoned description). The agent should not invoke.
REVIEW	decision	The eval ran but produced ambiguous or partial findings. The agent should defer to a human or fall back to its own checks.
UNVERIFIED	status	No verdict on file yet. Treat as not-yet-cleared (fail-closed). At launch, with the corpus at 15/150 (directory conformance labels, not the gate's drift corpus — see §5.6), this is the common live response.

Underneath the decision surface, the wire records *how strong the evidence is* on a separate clearance-level ladder (`VerdictGrade`: `unverified` → `scanned` → `crawl-l0` → `cred-l1` / `sandbox-l1` / `owner-authorized-l1`) so an `ALLOW` never reads stronger than its evidence; the ladder detail and the browsable directory surfaces (/screen, /stats, /leaderboard, /status, /changelog) are in §8.2. The screen half of the two surfaces is a concrete product, not only an API contract.

How a tool gets screened (the engagement loop, so the screen is something you can drive, not just receive). A tool moves up the clearance ladder three ways, all shipping code: an **owner claims and authorizes** their server (`ownership.py`) to unlock the higher owner-authorized evidence levels; a **crawler** screens published contracts at tiered evidence depth (`crawl.py`, the `crawl-l0` grade); and coverage is

demand-driven — the tools the most installs ask about are prioritized first (`demand.py`, `coverage_map.py`), adversarial cases ahead of the long tail. So the path from `UNVERIFIED` to a graded `ALLOW` is a queue a buyer can join (claim your server) or move (ask about a tool), not a black box.

For the modal launch buyer — wiring an unowned third-party MCP server they already use — the screen most often returns **UNVERIFIED** at launch (§5.6), so the concrete day-one deliverable is the **in-path tier-0 drift gate**, not a safety verdict on your existing tools. The "What runs for YOU on day one" box below — the load-bearing anchor for the held-state story; later sections cross-reference it rather than restate the held/empty state — makes that exact for each buyer segment, and the box after it makes the affirmative case for why the floor alone is worth the install.

⬡ **What runs for YOU on day one.** The modal buyer installs to protect tools they already use. Here is exactly what is live for them at install — and why "no banner" is the *healthy* state, not a broken one:

BUYER	TIER-0 DRIFT GATE	DIRECTORY VERDICT ON YOUR EXISTING TOOLS	TIERS 1–3	DAY-ONE OBSERVABLE
Solo dev	**LIVE** — pins + watches every tool now	most likely **UNVERIFIED** (§5.6)	held until you provision/activate	a `pin` →edit→ `check` HOLD you fire yourself + the "N tools pinned" stats card (§8.6)
Small team	**LIVE**, all postures, any seat count, free	most likely **UNVERIFIED**	held; activate the network later, opt-in	same; plus the per-server daily digest (empty = healthy)
Unowned third-party server	**LIVE** — drift-after-install caught going forward	**UNVERIFIED** (you do not own it)	tier-3 **DECLINED** (you cannot wire a verifier on a server you do not own) → flat HOLD	the pinned-inventory card; the first banner fires the first time the vendor's contract moves

Installing commits you to **nothing** about the network: the local gate is free forever, the network is opt-in later, and you lose nothing by installing now and activating the corpus when it has coverage you care about. On day one you get a **change-tripwire and a watched inventory**, not a clearance — and "nothing happened" is the gate working, quietly.

⬡ **Why the tier-0 floor alone is worth the install — the affirmative case.** The day-one floor is "just" a deterministic contract-diff, and a sharp buyer will ask whether that justifies an in-path proxy. It does, and here is the asymmetry in plain terms. The floor catches, *before* an irreversible call, the exact change classes that turn a trusted tool hostile: a silently-added **required** parameter, a flip to **destructive**, a **removed** parameter your agent relied on, a **narrowed** constraint or **reduced** enum. The cost of the floor is a free, always-on, zero-egress watcher; the cost of *not* having it is one un-caught destructive flip on a tool your agent already trusts — and your agent acting on it. That trade is lopsided. And in-path enforcement is not a diff you could comfortably hand-roll: it **persists the baseline across restarts** (Monday-to-Tuesday, §4.2), it **HOLDs the call before it leaves the client** rather than reporting after the fact (a passive sniffer can only alert), it **scans the output schema and description for injection/exfil markers** the structural diff alone would miss, and it runs through **one shared gate** reused verbatim across proxy and SDK so there is no second copy to drift. The floor is the product on day one; the network is what it grows into.

⬡ **The franchise.** The trust-to-act layer: it pins every tool's contract and holds the call the moment that contract moves — and across installs, that judgment is recognized everywhere rather than re-derived on each. The cross-install verdict corpus, the outcome-flywheel calibration riding on it, and the governance an ecosystem embeds against are the network the single gate queries (tier-1) and feeds. The gate is how one install starts; the layer is the moat (§6). The gate is copyable; the layer is not.

⬡ **Two surfaces, two deliberately different alphabets.** The screen and the gate answer the same question at two moments, but they speak different vocabularies, and a screen `ALLOW` is **not** a gate `PROCEED`. The screen never enforces; the gate always does. Keep them straight:

SURFACE	OUTPUT VOCABULARY	WHERE IT LIVES	WHAT YOU DO WITH IT
Directory screen (the prior)	`ALLOW` / `DENY` / `REVIEW` / `UNVERIFIED`	**advisory**, out of your call path (§3)	a decision input *before* you wire a tool; never blocks a call
In-path gate (the live check)	`PROCEED` / `HOLD` / `INCONCLUSIVE`	**in-path**, on every `call_tool` (§4)	enforces: a `HOLD` stops the call before your agent acts

4. The architecture

Everything below answers one question at the moment your agent spends its trust: should it act on this tool call, right now — has the contract moved?

⬡ Platform at a glance (live at launch vs. provisioned)

The whole product shape on one screen: two surfaces, four tiers, five deployment forms, marked live-at-launch vs. held. The single read before the architecture prose; the box a buyer screenshots to explain mcpindex internally. **Read the "Live at launch" column literally:** what is genuinely live day-one is the tier-0 gate, tier-3-when-you-wire-a-verifier, and the five deploy forms — tiers 1 and 2 are built-but-held *activating* seams, not load-bearing for the day-one value. "Full platform" means the mechanism is all shipped and audited; the *populated* network accrues, it is not flipped on (§6 cold-start).

COMPONENT	FORM	LIVE AT LAUNCH	NOTES
Surface 1 — in-path drift gate	the wedge, the live check	**LIVE**	pins + re-checks every call; HOLDs on drift (§3, §4)
Surface 2 — directory screen	advisory prior, before you wire	**LIVE (advisory), coverage 15/150**	reasons about a *published* contract; not in your call path (§3, §5.6)
Tier 0 — deterministic contract-diff	every call, zero egress	**LIVE**	the day-one floor; no model, no network (§4.1)
Tier 1 — cross-install corpus lookup	on a tier-0 INCONCLUSIVE	**HELD + EMPTY** (mechanism built; seam off, corpus empty)	**wired on the gateway path; the stdio proxy + SDK ship tier-1 UNWIRED** (`tier1=None`) — the modal buyer escalates a tier-0 INCONCLUSIVE straight past tier-1; activates when an operator provisions the seam (§5.2)
Tier 2 — cost-capped LLM consult	on the genuinely ambiguous	**HELD** (abstains, zero egress)	operator-provisioned model only (§5.3)
Tier 3 — behavioral verifier	on a behavioral-mandated INCONCLUSIVE	**CONDITIONAL** — runs only on your *own* OSS / owner-authorized server	proxy/SDK only; the modal unowned third-party server is **DECLINED → HOLD** (§4.1). The bare **LIVE** token is reserved for rows that are day-one for *everyone*
Deploy — stdio proxy	persistent pins	**LIVE**	the across-restart floor on the persistent `~/.mcpindex/pins/` path (§4.2)
Deploy — SDK `wrap()`	custody-free library	**LIVE**	one line around a session your host already authenticated (§8.2)
Deploy — one-click install	`install.sh` / `.mcpb`	**LIVE**	auto-wires hosts with consent; reversible (§8.2)
Deploy — HTTP gateway	remote upstreams, transit-only auth	**LIVE, loopback-only, fixture-tested**	the least-mature surface; ephemeral per-process pin, no persistent floor (§8.2, §9 #9)
Deploy — enterprise multi-tenant	logical isolation primitives	**HELD (flag off); single-tenant is the launch default**	spoof-resistant identity + VM separation buyer-funded (§6, §8.2)

4.1 The trust ladder: tier-0 → tier-3

For the modal launch buyer the ladder degrades to its tier-0 deterministic floor by design: tiers 1–3 are fully-wired seams that activate as the network, an LLM endpoint, or a behavioral verifier is provisioned. The floor is the product on day one; the upper rungs are the network the install grows into. mcpindex resolves a tool call through a single decision path with four tiers. Each tier escalates only what the tier below could not resolve, and **fails closed** at every boundary: the honest direction of failure is *stay held*, never *fabricate cleared*.

One dependency governs the whole ladder: pin-baseline integrity. Every rung diffs against the pin, so a surface that loses its persistent baseline starves *all four* tiers, not only tier-0 — a corpus lookup keys on a hash the gate computes only against a real baseline, so tier-1 coverage cannot rescue a gate that re-pinned the drift. This is why the ephemeral-pin HTTP gateway's restart-re-TOFU (§4.2) is a whole-ladder hole, not a tier-0 one; the operating rule is §8.2.

TIER	NAME	WHAT IT DOES	COST / WHEN IT FIRES	FAILURE DIRECTION
0	Deterministic contract-diff	Diff the client's current tool view (refreshed on re-list, §2) against the pin, classify into the `ChangeKind` taxonomy, scan for markers (a **trip-wire for known unobfuscated patterns, not an injection defense**, §4.3). No LLM, no untraceable score.	Every call. A pure in-memory diff: no I/O, no network, zero egress, sub-millisecond on the common path. Resolves any change it can classify locally.	An unclassifiable structural surprise → `deep-schema-undiffable` → HOLD, never silently "no change".
1	Cross-install corpus lookup (the flywheel)	On a tier-0 INCONCLUSIVE only, ask the corpus: *does a verdict exist for this exact public-contract hash?*	Only on a tier-0 INCONCLUSIVE. **Wired on the gateway path** (a local verdict cache first, then the held cloud seam); the **stdio proxy and SDK ship tier-1 unwired** (`tier1=None`, `proxy.py` / `preflight_intercept.py`), so the modal buyer's tier-0 INCONCLUSIVE escalates straight past tier-1. The cloud seam is **held by default and empty at launch** (§5.2). Only the `definition_hash` ever leaves, opt-in.	Fail-closed, two guards. Any error, or no tier-1 rung on this surface → a miss → the INCONCLUSIVE stays held. Where tier-1 *is* wired, the resolution runs through the shared `_tier1_adapter` **M1 guard** (`proxy.py`; the gateway routes its resolver through the same adapter), which **refuses a corpus `cleared` for any behavioral-mandated kind** (`annotation-flip-to-destructive`, `output-schema-changed`) and falls through to tier-3 — because the `definition_hash` excludes the very fields those kinds live in, so a stale clearance would otherwise ride the drifted hash. A corpus `dangerous` still reinforces a HOLD. This is the **sibling of the §5.3 tier-2 benign-downgrade guard**: both refuse to relax a behavioral-mandated INCONCLUSIVE to PROCEED. The *residual* poison surface is a corpus `cleared` on a **purely-structural** INCONCLUSIVE —

TIER	NAME	WHAT IT DOES	COST / WHEN IT FIRES	FAILURE DIRECTION
				currently empty by the same routing invariant §5.3 invokes for tier-2 — which is why activation still ships gated on the §10 evidence-standard change (§5.2).
2	Cost-capped LLM consult	On the genuinely ambiguous, a bounded LLM read of the description and schema, treated as hostile input. **Default `HeldLLMJudge`: abstains, zero egress.** At launch it can only **reinforce a HOLD or abstain** — the benign-downgrade-to-PROCEED branch is unreachable (every live INCONCLUSIVE is a Guard-dangerous kind), so tier-2 never clears a call today.	Off the hot path; budget-capped. Held by default; a real model is operator-provisioned (§8.1, §10).	An unavailable, abstaining, or over-budget consult does not unblock. An *opinion*, never a clearance: escalate-only on the live path.
3	Synchronous behavioral verification	The "third door": for one tool in an INCONCLUSIVE state, does its *actual* behavior match its *declared* contract? *Offered* on a behavioral-mandated INCONCLUSIVE; only *runs* when an isolator or owner-authorized read-only probe is wired, else UNAVAILABLE → stay HOLD.	Off the hot path.	Clears or refutes only on an observed battery; every absence, refusal, or error → UNAVAILABLE → stay HOLD.

⬡ **Which rungs run at launch.** Tiers **0** (always) and **3-when-wired** execute at launch; tiers **1** (cloud) and **2** (LLM) ship as built-but-held seams (`HeldCloudTier1Client` / `HeldLLMJudge`). The ladder is fully wired, but two rungs are no-ops until provisioned (§8.1 matrix). **Tier-3 is reachable only on the proxy/SDK path with an OSS or owner-authorized server; the HTTP gateway against an unowned upstream is DECLINED → flat HOLD** (`choose_behavioral_method`). For the **modal launch buyer** (an unowned third-party server, cloud and LLM seams held), tiers 1–3 are all unavailable, so a behavioral-mandated INCONCLUSIVE is a **flat HOLD routed to a human**. The ladder degrades to its fail-closed floor *by design*: a held rung never becomes a PROCEED. This is the canonical statement of the modal-buyer tier-0-only scope; later sections cross-reference here rather than restate it.

⬡ **The hot-path cost (illustrative; `calibrated=false` discipline).** For an in-path-on-every-call component the latency story is part of the design. On the **common path the contract is unchanged**, so the gate computes one SHA-256 over the canonical contract bytes and compares it to the pinned hash — a hash match short-circuits with **no structural diff at all** (the `ChangeKind` walk runs only when the hash differs). Tier-0 is a pure in-memory operation with no I/O, no network, and zero egress; the added latency on the hash-match fast path is **sub-millisecond, order-of-magnitude** (flagged illustrative, not a measured benchmark — we publish no calibrated number we have not earned, §5.5). The §8.1 local stats card surfaces *your* observed added-latency on your own traffic.

Every verdict's provenance (§5.4) records two axes: the **tier reached** (`tier_reached` , the deepest rung that *actually ran*, never one merely offered) and the **verdict scope** (`verdict_scope` : tiers 0/1 are `structural` , tier 2 is `semantic` , tier 3 is `behavioral`). A reader always knows what kind of evidence backs the decision. **Concretely for the modal case:** a behavioral-mandated INCONCLUSIVE that only *offers* the tier-3 check but runs nothing (no verifier wired — the modal buyer) records `tier_reached=0` / `verdict_scope=structural` , so the provenance never claims behavioral evidence that did not run. An INCONCLUSIVE that is *not* behavioral-mandated stays a flat HOLD; there is no path by which an unresolved INCONCLUSIVE silently becomes a PROCEED.

⬡ **tier-3 addressable audience, named once.** Tier-3 runs at launch only for a dev gating their **own** OSS or owner-authorized server; the modal buyer's unowned third-party tools are **DECLINED → HOLD by design**. And what a tier-3 CLEARED *attests* is scoped per leg: the OSS sandbox leg exercises behavior in a no-egress isolator, but the DEV_OWNED read-only leg attests **input-conformance without firing side effects** (`allow_side_effecting_baseline=False`) — so on a destructive-annotation flip, a read-only-leg CLEARED says "the declared input contract holds under probing," not "the tool's destructive runtime behavior was observed and is benign." The §4.1 "clears or refutes only on an observed battery" line is exact for the sandbox leg; for the read-only leg, "battery" means the input-conformance probes, not side-effecting execution.

> ⬡ **The honest box around the win (the spine reader's threat model, in 8 lines).** Now that the mechanism is on the page, here is the box around it — the full perimeter is the §2 security band. **What the gate defends:** any *change* to a pinned tool's contract after your agent trusted it — a structural input-schema drift, a destructive-annotation flip, a description edit, an injection/exfil marker, or a brand-new un-pinned tool. **The adversary** controls every field of a tool's self-description and the timing of any change, and can *withhold* the `list_changed` notification. **The one residual that matters at the headline:** the in-session catch requires the host to re-list; against an adversary who withholds `list_changed` (a **TOCTOU** seam), in-session detection is not guaranteed today — the floor is **next-launch detection from the persistent pin** (§4.2), and the in-session gap closes only when forced periodic re-list ships (§10). **What is out of scope** (a *change* detector, not a malice oracle): drift that predates the pin, a hostile tool that never changes (the screen's job, §3), and response-content injection (§9 #4). The five-case out-of-scope set, the per-host re-list table, and mcpindex's own attack surface are the §2 / §2.5 / Appendix security band.

4.2 TOFU pin + persistent baseline

On first sight of a tool (the client's `tools/list`) the gate pins the contract **trust-on-first-use**: a SHA-256 over `name` + `description` + `inputSchema` (canonicalized — see §5.2), plus the captured schema. The first-seen contract is the baseline; this is also the honest limit (§9 #2: the gate cannot catch drift that happened before it was installed).

Pin persistence is **per-surface**, stated exactly:

- **The stdio proxy and any SDK wrapper given a pin-store path persist across restarts.** The shipping proxy writes to `~/.mcpindex/pins/<server_id>.json`, loads the prior baseline on launch, and persists every TOFU/re-pin atomically (mkstemp + `os.replace` whole-file rewrite), so a contract that changes while your agent is offline is **caught on the next launch (code-readable today in `proxy.py`: `default_pin_store_path` → `~/.mcpindex/pins/`, atomic write, load-on-launch; the runnable across-restart rig is the near-term §10 leg).** This is the property behind the Monday-to-Tuesday promise, **on the persistent stdio/SDK path**. The within-session §7 rig already runs through this persistent path; the across-restart harness is the natural next leg, not a different mechanism.
- **The HTTP gateway currently runs an ephemeral, per-process pin — it has NO persistent detection floor today.** It re-TOFUs on every restart; cross-restart persistence on the gateway is a deploy-held seam (§10). The threat consequence is explicit: a restart coincident with drift (a deploy, a crash, an autoscale event) re-TOFUs and silently accepts the drifted contract as the new baseline. That is exactly the mutation-after-trust the product exists to catch, defeated by a restart on the remote path — and unlike the stdio proxy (whose persistent pin gives a next-launch floor), the gateway has *no* detection floor below the in-session one. The operator decision rule is in §8.2: for any remote server where drift-across-restart matters, use the persistent-pin stdio proxy.

4.3 The deterministic ChangeKind diff

The precision core is a pure, deterministic, side-effect-free classifier (`schema_diff.py`): no I/O, no network, no clock. Given the before/after self-descriptions, it walks the JSON-Schema structure and maps every delta into a fixed **14-member taxonomy**:

- **Breaking:** `added-required-param`, `removed-param`, `type-changed`, `enum-values-removed`, `constraint-narrowed`, `required-set-expanded`, `tool-removed`.
- **Behavioral-mandated:** `annotation-flip-to-destructive` (`destructiveHint` gains `true`, OR a previously-held `readOnlyHint` is dropped) and `output-schema-changed`. Annotations are *spec-*

untrusted hints, which is precisely why an annotation flip cannot clear or condemn on the declaration alone and is routed to behavioral verification (§4.4): trusting the declared bit would be trusting an attacker-controlled field.

- **Additive / low:** `added-optional-param`, `output-schema-added`, `tool-added`.
- **Cosmetic:** `description-only` (routed to a dedicated description-poisoning HOLD, the most attacker-controllable channel).
- **Fail-safe:** `deep-schema-undiffable`: a schema too deep (or `$ref`-expansion-cyclic) to fully diff joins the must-review list rather than passing as "no change". The recursion bound is `MAX_DEPTH = 16`; an attacker-sized schema cannot blow the stack or pass silently.

A single `safety_relevant` boolean is derived *from* the taxonomy, in exactly one place, so the bit can never disagree with the kind. The classifier also unions the *effective* `required` set across `allOf` / `anyOf` / `oneOf` branches: a parameter made required only inside a combinator branch is still required to call the tool, and the gate fails safe by treating it as required.

Three precisions on what the diff is and is not.

First, a description rewrite is caught as a *change* (the byte-exact description-poisoning HOLD), and the input/output schema and description are scanned for known injection/exfil **markers**. The deterministic gate does *not* semantically classify a first-seen description as a poisoning payload; that semantic judgment is the screen's (§3) and the advisory tier-2's (§5.3).

The marker scan is a trip-wire, not an injection defense. It matches **known, unobfuscated patterns** (imperative-instruction substrings and known-exfil-sentinel substrings — the class, not the published list). An attacker who controls the bytes encodes around it trivially. We treat it as a low-cost early-warning, **not** an injection defense; the real defensible claim is the byte-diff: a poisoned description is caught as a *change* regardless of whether the scanner recognized the marker (§9 #3).

Second, the gate scans for markers not only in the input schema and description but in the **`outputSchema`**, an unhashed, agent-consumed channel excluded from the canonical invocation hash. The output-schema marker leg is scanned *first* and is independent: it fires even on a hash match, before any structural decision.

Third, the classifier compares `$ref` reference *strings*, not their resolved targets, so a `$defs` body rewritten under a stable ref name is invisible to the *structural classifier*. But the canonical pin hash (§4.2) is over the full `inputSchema` bytes (`$defs` included), so a rewritten `$defs` body changes those bytes, changes the `definition_hash`, and trips the TOFU mismatch. The leg is mechanically traceable in code: on a hash mismatch the classifier returns an empty structural diff, and the gate **fail-closes that empty-diff-on-hash-mismatch to a synthetic `deep-schema-undiffable` HOLD** rather than auto-clearing (`gate.py`), so the change HOLDs even though the classifier cannot name the kind (**within a single implementation** — the cross-language byte-identity premise of §5.2 governs the backstop's strength, so the two limits are one perimeter). A `$defs` edit that canonicalizes to *identical* bytes (pure key reordering) is by design not a change and correctly produces no HOLD. Defense-in-depth, not a silent miss — and the one way to defeat it is a same-privilege process that rewrites *both* the upstream `$defs` and the stored pin, which is already inside the §2 / §8.4 compromised-host boundary (one perimeter).

4.4 Postures: Monitor / Guard / Strict

The same diff feeds three operator-selectable postures. Posture is per-install config; the default is **Guard**. A behavioral-mandated INCONCLUSIVE is not a softer PROCEED; under Guard it HOLDs the call until behavior is observed.

- **Monitor**: never blocks. Every call PROCEEDs; a safety-relevant drift is recorded and surfaced as a notify-only note. The break-glass posture for an outage, and the recommended first-week posture for sizing your own HOLD rate before switching to Guard.
- **Guard (default)**: the production posture. It **HOLDs** the unambiguously breaking kinds and any injection/exfil marker; it **routes the behavioral-mandated** kinds to INCONCLUSIVE, with run-behavioral auto-offered *when a verifier is wired*. Until behavior can be observed, the call does not proceed. (For an unowned third party, where no verifier can be wired, the INCONCLUSIVE is a flat HOLD with a behavioral-unavailable note.) It lets a *structurally-benign* drift through. An un-checkable tool (no pin, derivation error) HOLDs.
- **Strict**: holds on *any* safety-relevant drift; the behavioral-mandated kinds still surface as INCONCLUSIVE. An un-checkable tool HOLDs.

> ⬡ **Living with the gate — and the shape of what HOLDs.** A trust gate that cries wolf gets uninstalled in a week, so friction is part of the design, and the friction is **predictable from the taxonomy** before you measure your own number. What **auto-passes silently, no banner**: the additive classes — `added-optional-param`, a brand-new additive tool (`tool-added`), an additive first-time `output-schema-added` — plus a contract that canonicalizes byte-identical (no change). What **HOLDs**: the breaking set (`added-required-param`, `removed-param`, `type-changed`, `enum-values-removed`, `constraint-narrowed`, `required-set-expanded`, `tool-removed`), the behavioral-mandated set (`annotation-flip-to-destructive`, `output-schema-changed`), a `description-only` rewrite (the most attacker-controllable channel), and any injection/exfil marker. **A typical SemVer-respecting vendor minor release falls in the silent-pass set** — it adds optional surface, not breaking changes — so it passes without a banner. *Illustrative dogfood trace:* in the §7 battery, `added_optional` (a vendor minor bump that adds an optional param) is **auto-accepted and re-pinned, no banner**; only `added_required` (a new *mandatory* param) HOLDs. When a HOLD does fire, recovery is one action: **Re-pin** accepts the new contract as the baseline and the next call proceeds, sub-second, one click in the client. If your vendors push frequent doc edits, run Monitor and review the daily per-server digest (`cse/digest.py`, §8.6) rather than holding each; an empty digest is a healthy state. **For your exact number:** the taxonomy gives the *shape* of what holds; run Monitor for a week against your real servers and read the local stats card (§8.1, zero egress) to get your precise rate before switching to Guard.

4.5 HOLD-before-call and the ⬡ banner

A HOLD is a refusal **before** the call leaves the client. On the JSON-RPC wire, the proxy returns a held error that is *not forwarded upstream*; the tool is never invoked. The fail-closed surface is honest about *which* kind of refusal it is, on three distinct codes so an internal bug can never masquerade as a drift verdict:

- `-32010`: a **contract-diff HOLD** — the contract drifted versus your pin.
- `-32011`: an **upstream/transport failure** — the gate could not reach the server, or the upstream frame was unreachable, unparseable, or over the body cap.
- `-32012`: an **internal error in mcpindex's own code**, failed closed under its own distinct code (never dressed up as a drift verdict, never carrying a stack trace or user data).

The surfaced moment for a contract-diff HOLD is a deliberately calm brand beat:

> ⬡ mcpindex — caught a silent change: make_report now requires a new parameter (owner): existing calls that omit it will fail. Held before your agent ran it. [Review · Re-pin · Validate]

The flow is concrete: the agent attempts the call → the gate catches the drift and surfaces the ⬡ banner *in-client* with the before/after contract diff → you choose **Review**, **Re-pin** (accept the new contract; the call proceeds), or **Validate** (run the behavioral check, where wired) → one click resumes. "The human decides" is a button, not a slogan. Every attacker-derived field interpolated into the banner is control-byte-stripped and green-word-redacted (§2.5), so a tool named `all clear verified` cannot smuggle a fake "safe" line. The voice is *"a silent change was caught,"* never *"this tool is unsafe."*

4.6 Credential handling, scoped per surface, and zero body logging

The gate's relationship to your credentials is **structural, not a promise**, scoped per surface. Credentials are never *stored*; the one thing that *is* stored on-host is the pin, which carries no secret and inherits the user's file-permission model (§8.4).

- **The SDK wrapper is custody-free.** It has no token field, no auth parameter, and never opens its own connection. It wraps a session the host *already authenticated* and calls only that session's `list_tools` / `call_tool` / `send_request`. The credential lives entirely inside the session the host owns. There is nothing to steal because there is nothing to hold.
- **The HTTP gateway is transit-only.** It threads the host's existing auth header through to the upstream per request, never baked in, never persisted, never logged. The credential transits the process; it is never stored on any surface.

The exact line: **mcpindex never *stores* a credential on any surface.** Alongside it: **no request or response body is ever logged or persisted** (a body could carry a secret, so the gateway logs no body, bounds every body, and fails closed on an over-cap or unparseable one). The **stdio proxy applies the same discipline**: it never logs a frame body, only closed-vocabulary breadcrumbs. This never-logged guarantee is **verified by code-read across the proxy, gateway, and connector modules** (no header or body in any log call); a negative-assertion test that greps emitted telemetry for credential/body bytes is the runnable backstop (§10). (A body does transit to the legitimate authenticated upstream, the proxy's function; what is guaranteed is never-logged, never-persisted, never-sent-to-any-mcpindex-sink; see §8.1.)

> ⬡ **The in-path blast radius, named once.** The gate is a new in-path component that *observes* every bearer token in transit for every gated upstream. "Never stores a credential" is true and verified, but it is not the only question a CISO asks, so here is the whole answer in one place. **Blast radius if the gate process is compromised — single-tenant default:** transit-only · never-logged · loopback-bind by default (the gateway) · user-privilege, not root · single-tenant by default. **On a shared multi-tenant gateway the blast radius widens:** a compromised gateway owns **N tenants' credentials in transit** at once, consistent with the transit-only model — which is exactly why hardened multi-tenant separation (spoof-resistant identity + per-tenant process/VM isolation) is buyer-funded, not the launch default (§6, §8.2). **What does NOT reduce it:** a popped gate process owns the live tokens for its own session, the same as *any* in-path proxy — that is the deliberate trade of putting a gate in-path, not a defect unique to mcpindex. (Credentials are never stored; the pin store *is* stored on-host and inherits the user's file-permission model — see §8.4.)

5. The methodology

§4 is the shape of the system; §5 is the discipline that keeps a change detector from masquerading as a safety oracle.

5.1 Deterministic: the structural floor

Tier-0 is the floor and the most trustworthy layer, precisely because it is the least clever: a pure structural diff with a fixed taxonomy and a documented fail-safe. No model in the loop, no score you cannot reproduce. The dogfood proof (§7) runs the scenario battery against the *live gate code*; the gate is ground truth, not a hand-written table. Tier-0 correctness is **definitional, not empirical**: a question of whether the diff is faithful, not of a detection rate. (Calibration, §5.5, therefore scopes to the semantic and behavioral tiers, not tier-0.)

5.2 Corpus / flywheel: the mechanism is built and live; the corpus is empty at launch

Tier-1 fires **only on a tier-0 INCONCLUSIVE**: the deterministic floor resolves locally any change it can classify, with zero egress; only the narrow behavioral-mandated classes escalate. **Tier-1 is per-surface, and the modal buyer has no tier-1 rung at all today:** the local verdict cache and the tier-1 resolver are wired only on the gateway path (`gateway.py`); the stdio proxy and the SDK ship tier-1 **unwired** (`tier1=None`, `proxy.py` / `preflight_intercept.py`), so a tier-0 INCONCLUSIVE on the stdio/SDK path escalates straight past tier-1 to the held tier-2/tier-3 seams — a flat HOLD for the modal buyer. This *strengthens* the cold-start honesty: the day-one stdio/SDK install is the tier-0 floor, full stop, with no corpus rung in the path until the gateway and the cloud seam are both in play. A tool's **public-contract hash** is cross-install-deterministic: the canonical form is a SHA-256 over a normalized JSON serialization (sorted keys, fixed number/whitespace/unicode form), so the same contract produces the same hash on every machine and a verdict the corpus learned for that hash resolves the *same* contract for every other install.

> ⬡ **The corpus MECHANISM is built and live; the corpus DATA is empty at launch.** The schema is fixed, the wire contract is frozen, the lookup and contribute code paths ship — but the cloud seam ships **held** (`HeldCloudTier1Client`, zero-egress, always a miss) and the contribute path ships **off by default**, so the corpus is **empty on the day-one install**. The flywheel turns the moment an operator activates the seam (which is gated on the §10 evidence-standard change below, not flippable at launch), not on the default install. This is the bible's (A)/(B) split made precise: the mechanism is deploy-held → present-tense; the *populated, calibrated* network is maturity-held → it accrues, it is not flipped on. The network compounds from empty.

Hash determinism — the load-bearing premise and a stated limit, scoped to its true fail-direction. Cross-install hash equality is the floor of the flywheel. The narrow residual is a small set of edge encodings (e.g. integral-valued floats) where the Python gate and the TS port could canonicalize a byte differently; a mixed Python/TS shop should not yet assume shared cross-language coverage. **Crucially this limit fails safe in one direction only:** a canonicalization mismatch makes the two ports compute *different* hashes for the same contract, which degrades to a tier-1 cache **MISS** — the INCONCLUSIVE stays held — never a wrong PROCEED. So the limit costs coverage (a corpus verdict one port learned may not resolve on the other), not safety. A §9-class limit, not an assumption that the ports silently agree.

The egress discipline is strict: on a lookup, the *only* thing that leaves is the `definition_hash` — no arguments, no schema text, no credential, no tool name. The contribute path sends only the anonymized

triple `{definition_hash, verdict, reason_class}`, where `reason_class` is validated against a closed vocabulary and dropped to empty on any violation. The cloud client reuses the same HTTPS-only, public-IP, no-redirect, IP-pinned transport as the upstream connector, so even a misconfigured or rebound cloud URL cannot reach internal/metadata addresses.

One scope precision: **a hash-keyed corpus verdict transfers a CONTRACT-level judgment across installs, not a behavior-level safety clearance.** Identical bytes can front different runtime behavior (§2). The corpus accrues "this exact contract has been judged"; it cannot accrue "this contract is safe to run."

○ **The canonical tier-1 safety home (grep-verified against `proxy.py` / `gateway.py` / `preflight.py`; every other location cross-references here).** The real `CloudTier1Client` is *never* the default (a load-bearing safety property) and refuses to construct without an operator-supplied base URL and bearer key. The tier-1 wire contract (`Tier1Verdict`) is intentionally minimal: `{definition_hash, verdict ∈ {cleared, dangerous}, reason_class}` over a closed vocabulary, with **no tier, no scope, no clearance level, no expiry** — so a cloud bit arrives with strictly less evidence-metadata than a locally-produced verdict.

A `cleared` for a behavioral-mandated drift is already refused in code — the M1 guard. The effective shared resolution path is *not* `gateway.py` `_to_resolution` in isolation; both the stdio proxy and the gateway route the resolution through the proxy's `_tier1_adapter` (the gateway passes its `Tier1Resolver.resolve` as the core's tier-1 callback, `gateway.py` `build_route`). `_tier1_adapter` carries the **M1 fail-closed guard**: when the static drift carries *any* behavioral-mandated kind (`annotation-flip-to-destructive`, `output-schema-changed`), it **refuses to relax a corpus `cleared` to PROCEED and returns `None`**, forcing tier-3 escalation. The reason is exact: `definition_hash` is `_tool_hash(name, description, inputSchema)` and **excludes `outputSchema` + `annotations`** (`preflight.py` `_hash_tool`) — the very fields the two behavioral-mandated kinds live in — so a `cleared` minted for the *original* contract shares the *drifted* contract's hash, and the guard exists precisely to stop that stale clearance from waving a read-only→destructive flip through. A corpus `dangerous` still reinforces a HOLD. This is the structural **sibling of the §5.3 tier-2 benign-downgrade guard**.

So the poison vector is narrower than a blanket "unconditional PROCEED" reading. The wave-through the §2.5 #4 vector describes — a malicious contributor seeds a favorable `cleared` for a hash it controls, then ships hostile behavior on that contract — is **already closed in code for the highest-stakes drift classes** (the destructive flip), because the M1 guard refuses `cleared` on every behavioral-mandated kind. The *residual* poison surface is a corpus `cleared` on a **purely-structural** INCONCLUSIVE. That residual is **currently empty by the same routing invariant §5.3 invokes for tier-2**: every INCONCLUSIVE-eligible kind the gate emits today is also a Guard-dangerous (behavioral-mandated) kind, so no purely-structural INCONCLUSIVE reaches the resolution point. At launch the vector is *doubly* closed — the seam ships **held and empty** and contribute ships **off**, *and* the M1 guard refuses the destructive subset even if it weren't.

What is still owed at §10, stated precisely (not "no guard exists"). The M1 guard closes the behavioral-mandated subset; what it does *not* yet do is enforce a general re-derive-locally evidence standard at the resolution point for a future purely-structural INCONCLUSIVE class — so if a non-Guard-dangerous INCONCLUSIVE kind is ever added, a corpus `cleared` *would* resolve it to PROCEED-with-note without local re-derivation. That general evidence-standard asymmetry — a cloud `cleared` resolves only by triggering a *local* re-derivation (a tier-2 consult and/or a tier-3 behavioral check on the dev-owned path; the corpus bit becomes a routing hint that selects *which* local check to run, never a verdict that substitutes for one), while a `dangerous` still condemns — must land in the **same change** that turns on contribute/lookup (§10). Until then the seam stays unactivated, which is the launch default. The precise claim, stated so a grep cannot falsify it in either direction: the M1 guard ships and closes the destructive subset today; the general structural-INCONCLUSIVE evidence standard is the §10 work — *not* "there is no shipped guard," and *not* "a corpus `cleared` resolves unconditionally."

mcpindex operates the canonical cloud corpus; a self-hosting operator can run their own. When the seam is enabled, the canonical mcpindex corpus runs on **Vercel + Supabase (US-region at launch; EU-residency is a §10 item)** as named sub-processors. At launch the canonical corpus is **held and empty by design** (above, §9 #8) — the cloud backend is provisioned in lockstep with seam activation, not standing

idle. The only datum that transits is the `definition_hash`, and a DPA / sub-processor disclosure attaches at enable time.

The cold-start truth: the network begins empty, and the contribute path ships held by default. Install #1's value is the local tier-0 gate, complete and zero-egress on day one; the network is the part that compounds (§6).

5.3 LLM consult and behavioral verification

Tier-2 is a cost-capped LLM read for the genuinely ambiguous. **The default is `HeldLLMJudge`: it abstains, makes no model or network call, and egresses nothing**, every INCONCLUSIVE escalating straight past it to tier-3. A real model (`ConfigurableLLMJudge`) is **injected only at the enterprise tier and never default-wired**; when provisioned, the *only* thing that leaves is a **bounded, curated contract-diff prompt** (the description plus a schema excerpt, capped at 4096 chars), never a credential, never tool arguments, never user data, sent to **the operator-configured model endpoint** (on-host or a cloud provider, the operator's choice and the single most important fact for a data-egress assessment). The description is hostile input: the judge cannot mint a positive clearance on the live path (escalate-only while `AUTONOMY_ENABLED = False`, §5.5), so a manipulated judge fails toward HOLD. A tier-2 resolution is always an **opinion**: advisory, never a deterministic or behavioral verdict.

The benign-downgrade branch ships off and is unreachable today. One knob, `tier2_allow_benign_downgrade`, gates whether a `LIKELY_BENIGN` opinion may resolve a structurally ambiguous INCONCLUSIVE down to PROCEED. It ships **OFF** and is currently **unreachable** by an explicit routing invariant: **every INCONCLUSIVE-eligible kind the gate emits today is also a Guard-dangerous kind** (the behavioral-mandated set), so no benign INCONCLUSIVE reaches the downgrade branch and it never fires. "LLM robustness is not load-bearing on PROCEED" is therefore true today *by that routing invariant*, not by the LLM's good behavior. Turning the flag on for a future non-dangerous INCONCLUSIVE class *would* make tier-2 adversarial robustness load-bearing on PROCEED: a deliberate, reviewed change, not a default.

Tier-3 is the **behavioral verifier**, the third door. Given one tool whose declared contract drifted into an INCONCLUSIVE state, it asks whether the tool's *actual* behavior matches its *declared* contract — and what "behavior" means is scoped per leg: **the OSS sandbox leg can exercise behavior** (it spawns the tool in a no-egress isolator), while **the DEV_OWNED read-only leg observes input-conformance / refutes a reachable declared-contract violation *without firing side effects*** (`send_malformed_probes=True, allow_side_effecting_baseline=False`) — it does not observe a side-effecting tool's full runtime behavior. It maps an observed clean battery to **CLEARED**, an observed violation to **FAILED**, and *everything else* (no isolator wired, an unauthorized tool, a non-read-only tool, any spawn or transport error) to **UNAVAILABLE**, which stays HOLD. It never runs untrusted code without no-egress containment, never fires a side-effecting tool, and never raises into the gate. It is *offered* on a behavioral-mandated INCONCLUSIVE but only *runs* when an isolator (for OSS the dev can spawn) or an owner-authorized read-only probe (for the dev's own server) is wired; for an unowned third party it is DECLINED, and DECLINED, UNAVAILABLE, or absent stays HOLD.

A tier-3 CLEARED is a **dev-local, ephemeral** clearance for the dev's own agent on the dev's own machine, for the current run. It writes nothing to the corpus, mints no public grade, and carries no outward-facing attestation. **The behavioral tier clears or refutes a contract change; it does not prove the tool is safe** (§9 #7).

The behavioral probe at v1. In the directory's eval pipeline the conformance probe is **monitored, not enforced**: a conformance failure surfaces in the verdict; it does not block the call upstream. Enforced

positive clearance is a separate, gated one-way door (§10).

5.4 Provenance and the over-claim guard

Nothing surfaces without a **complete provenance record**, enforced by a fail-closed choke (`require_provenance`). The record carries four immutable layers: the subject (which tool, hash old→new), what fired (the `ChangeKind`(s), `tier_reached`, the verdict, `verdict_scope`), the bounded before/after evidence (re-derivable), and an **honest-framing tag** (`contract-diff-not-safety`, `behavioral-needed`, `visibility-not-blame`, or `healthy`) encoded so a renderer can never silently upgrade a contract diff into a safety claim. When provenance cannot be built, the HOLD still stands, but the message degrades to a generic "held (provenance unavailable)" notice.

5.5 Calibration honesty

We publish no confidence number we cannot defend. **`calibrated=false` until calibrated against a held-out corpus.** The calibration loop runs in **shadow / measurement mode**: a module-level `AUTONOMY_ENABLED = False` kill switch. While False, the live decision path always escalates; it pins escalate-everything semantics regardless of any calibration passed in, so the empirical store can never mint a positive clearance on a live call. The store is fail-closed and tighten-only: insufficient data → P(wrong) = 1.0; an unknown source or a tampered ledger → an empty store → escalate-all. The shadow report computes a **reliability (calibration-error) score** over sufficient cells — an observation-weighted mean-squared gap between predicted P(wrong) and the per-cell empirical rate (a Murphy-decomposition reliability component, not the per-outcome Brier score), lower is better-calibrated — reported as n/a while every cell is fail-closed at P(wrong)=1.0 today: a real calibration instrument with no unearned number.

> **Naming artifact (grep-true, flagged so an auditor lands on it immediately).** The shipping code still labels this field `brier`; the quantity it computes is the **reliability component** described above, not the per-outcome Brier score. The field rename to `reliability` / `calibration_error` (or fixing the `render_shadow_markdown` label string) is a code-only fix on the §10 list — disclosed here rather than discovered mid-grep. A large reliability gap under the current fail-closed prior — predicted P(wrong)=1.0 against a low observed wrong-rate — is the **expected signature of a deliberately conservative gate, not evidence of miscalibration**: the store is pinned to escalate-everything, so the gap is by construction, and it closes only when a held-out corpus earns a real number. Flipping autonomy on is a separate, deliberate, reviewed one-way door.

> ⬡ **The live outcome stream can calibrate the held direction; it cannot certify the cleared/proceed direction.** The held-out adversarial corpus is the only admissible ground truth for CLEARED-class calibration. The reason is statistical: "cleared-and-fine" is **censored / right-truncated data** — the *absence of a reported incident*, not an observed clean outcome — and a false CLEARED is precisely the failure whose harm is least likely to be reported back as a labeled incident, so estimating P(wrong | CLEARED) from absence-of-complaint under-counts exactly the errors that matter. The full three-source independence analysis (and the HOLD-fatigue contamination of the human-override label, §2.5 #5) is in the Appendix; the load-bearing claim is the bolded line above.

5.6 What we have measured, and what we have not

This is an eval-methodology paper, so it states its own evidence exactly. **The headline up front:** at launch we have **fidelity evidence** (the §7 dogfood proves the wire faithfully carries each verdict) but **zero detection-power measurement** — no recall, false-positive, or precision number over a labeled

adversarial *drift* population, because the held-out adversarial drift corpus is **not yet built** (§5.5, §10). Everything below specifies exactly which number is which, and which corpus would earn it.

Structural possibility, not measured prevalence. We argue the threat from the protocol's structure and the disclosed attack classes [1]–[5]; we do **not** claim a measured in-the-wild drift incidence rate, and the MCP-specific disclosures [1]–[3] rest heavily on one vendor's reports (Invariant Labs), with [5] (the MCP spec's own *Security Considerations*) the independent corroboration of the structural premise. The held-out adversarial drift corpus (§5.5, §10) is the instrument that would establish detection power against real drift; until it is built, prevalence is asserted from structure, not evidenced from incidents. The tier-0 catch itself is **definitional, not statistical** — a question of whether the diff faithfully classifies a change, not of a detection rate (§5.1).

> **D3 graduation gate (the directory graduation gate on `/methodology`).** ≥150 conforming labels with the false-positive rate bounded as below. **Current: 15/150.**

Two corpora, not one. The 15/150 is the **directory screen's conformance-label graduation** (the *publish*-time prior, §3). It is distinct from the **held-out adversarial drift corpus** that would calibrate the in-path gate's CLEARED direction (§5.5) — a separate corpus that is *not yet built*. The directory number does not transfer to the gate.

The acceptance criterion must be specified unambiguously, because an under-specified one is worse than none in a paper that polices its own rigor. The bound, stated with its tail: **a one-sided 95% Clopper-Pearson upper limit on the false-positive rate over the labeled conforming/negative class ≤ 2%.** Under that one-sided reading, the achievable N is *not* prohibitive: at 0 observed false positives, the one-sided 95% upper limit is `1 - 0.05^(1/n)`, which at n=150 is **1.98%, already below 2%**. So a clean 0/150 *satisfies* the FP bound; N=150 is sufficient for the specificity half if no false positive is observed. The reason to target a larger negative population (N ≈ 250–300) is **to tolerate ≥1 observed false positive** (at n=150 a single FP pushes the upper limit well past 2%), not because 0/150 fails.

Critically, an FP-only bound measures **specificity, not detection power**. The defensible gate is two criteria: the FP upper-95 bound over the negatives **and** a **recall lower-95 floor over a held-out adversarial-positive drift population** — directionally, an *aggregate* recall lower-95 ≥ 0.9 over ≥100 pooled held-out adversarial-positive cases (Clopper-Pearson). The tail cuts the same way the FP side does: a one-sided 95% lower bound ≥ 0.9 at n=100 tolerates at most ~4 misses (96/100 gives a lower-95 of ~0.90; 95/100 already falls below), so the floor demands a **point recall near 0.95, not 0.90** — it is not "just hit 90% on 100 cases." The 0.9 floor is an aggregate over the pooled population; per-kind stratification (across the breaking + behavioral-mandated kinds) is for reporting and coverage, **not** a per-stratum confidence bound: ≥100 total cases cannot support a per-kind lower-95. **The aggregate can mask a low-recall safety-critical stratum** — a high pooled recall is consistent with near-zero recall on one dangerous kind, so the autonomy flip (§5.5) is gated additionally on *no observed miss in any safety-critical stratum*, regardless of the aggregate: an aggregate-only floor is necessary, not sufficient. That recall floor is on the roadmap (§10), gated on building the adversarial corpus, and is a target to earn, not a measured number.

The honest state today: **mcpindex publishes no recall, false-positive, or precision number over a labeled adversarial *drift* population yet.** The §7 dogfood is a **deterministic end-to-end fidelity test**, not a detector-quality eval: it proves the wire faithfully carries each verdict, not the recall/FP rate against adversarial real-world drift.

6. The moat: corpus × outcome-flywheel × governance

A contract judged once is recognized for that exact contract on every install, forever. The 10,000th install resolves changes the 1st could not. ("Recognized," not "cleared": a hash-keyed corpus verdict transfers a *contract-level judgment* across installs, never a behavior-level safety clearance, §5.2.) That compounding is the asset; the diff is the demo. **At launch the corpus mechanism is built and the schema is fixed, but the cloud seam ships held and the contribute path ships off — the flywheel turns the moment an operator activates the seam, which is gated on the §10 evidence-standard change (§5.2), not on the day-one default install.** The network is the part that compounds; it accrues, it is not flipped on. The network compounds along three separable axes, and the last one is the one a competitor stares at and gives up on.

1. Corpus — the coverage map. Every cross-install lookup keys on a public-contract hash, so the corpus is the shared map of *which exact contracts the world has seen*. Coverage, not judgment, is the axis.

2. Outcome-flywheel — the calibration loop. The loop from "we caught a change" to "we know how often that kind of catch was right." It accrues the **held/refute** direction now, with **deterministic ground truth (the §7 dogfood) as the clean held-direction signal** and the **human-override channel weighted, not trusted raw** — HOLD-fatigue contaminates the override label (Appendix). The cleared/proceed direction requires the held-out adversarial corpus, because "no complaint" is censored data (§5.5). You accrue it, not buy it.

3. Governance — the uncopyable axis. The honest-framing tags, the closed reason-class vocabulary, the verdict-scope axis, the OTS-anchored published history, and the open-core firewall encode a *governance posture* into the platform: the discipline that a verdict is a contract-diff, that evidence is re-derivable, that the moat cannot mint what the public surface is forbidden from minting. Framework partners and enterprises embed against that governance. A competitor can copy the diff in a weekend; it cannot copy the coverage map you have accrued, the calibration you have earned, or the governance an ecosystem has already embedded against.

The enterprise decision log. What is built today: an **in-memory, monotonic-`seq`, ring-buffered** per-tenant ledger (`TenantAuditLog`), closed-vocabulary, secret-free (an event-class enum, a monotonic sequence number, a wall-clock timestamp, and an optional public `definition_hash`; never arguments, schema text, or credentials). Tenant isolation is load-bearing: each tenant owns its log, and the monotonic `seq` lets an auditor see a gap-free ordering. It is **not** durably hash-chained today; the ring buffer evicts oldest-first by design. A buyer with a regulatory log-retention obligation should treat **durable SIEM export (§10) as a procurement prerequisite, not an enhancement**: the launch ring buffer is operational telemetry, not a retained audit record. A durably-chained, exportable SIEM tenant log with cryptographic non-repudiation is buyer-funded provisioning (§10), alongside the KMS signer.

Where tamper-evidence *is* real today is the published directory verdict history: HMAC software-signed at write time (a single-writer integrity tag, verifiable by the trust service, not third-party non-repudiation); third-party tamper-evidence comes from the OTS Bitcoin anchor any skeptic can recompute offline (the OTS pipeline detail is in the Appendix). KMS-signed non-repudiation is the §10 seam.

Commercial model (open-core). The **gate, SDK, and diff are free / source-available** (PolyForm-Noncommercial per the repo license). The **paid surface is the network and the enterprise tier**: the cross-install corpus / cloud tier-1, the verdict-minting that stays behind the moat (the open-core import firewall — `open_core_manifest.py` — is the governance mechanism that keeps the public surface from minting what the moat is forbidden to expose, the §6 governance axis above), and enterprise multi-tenancy.

TIER	WHAT YOU GET	PRICE AT LAUNCH
Local gate	Unlimited local gating, all postures, all hosts, zero egress: the full tier-0 contract-diff	**Free, and stays free** for individual and team use under the stated license
Directory API (Pro)	Higher rate limits on the directory/screen verdict API	**The single launch list price; the exact figure is set at go-live and published on /pricing (a known-open, not a number this doc withholds).** It is priced for the **agent-integration / programmatic-read use case** — high-volume polling of the four-state contract, including treating **UNVERIFIED as a fail-closed gating signal in your own CI / agent loop** (a coverage-independent, day-one value), **not** for verdict coverage (coverage grows with the corpus, §5.6). The standalone price is **transitional: it folds into Network at go-live**, so you pay for API access, never "for an empty directory" *(directory-API tier, distinct from the gate/network tiers)*
Network (team)	Cross-install corpus / cloud tier-1 lookup + contribute; metered on cross-install lookups	**Design-partner-set (§5.2):** the network mechanism is built but the corpus is empty and the cloud seam ships held at launch — this is a design-partner surface, not a day-one-purchasable populated network. Cross-language corpus sharing is single-implementation at launch (§5.2), so a mixed Python/TS shop should not assume shared coverage yet. Per-seat / per-lookup pricing set with design partners
Enterprise	Multi-tenancy* + the §10 provisioning bucket (real OIDC/JWKS, KMS-signed receipts, durable SIEM export, HA)	**Provisioning engagement:** buyer-funded; not flip-on-deploy

* **Multi-tenancy** at launch = cooperative, same-trust-domain tenants only (header-trust + logical isolation, ships OFF by default); spoof-resistant OIDC/JWKS identity + per-tenant process/VM separation is buyer-funded §10. See the §8.2 operating rule.

You only pay when you opt into the cross-install network or the enterprise tier; local gating is free at any seat count. The local gate — the day-one value — is and stays free. **Only the Directory API (Pro) tier carries a public list price at launch; Network and Enterprise are design-partner / engagement-priced** — so where a number is absent, it is design-partner-set, not missing by oversight.

> ⬡ **The cold start, owned.** A new entrant starts with an empty corpus and a cold flywheel; the network's value to install N+1 is the verdicts accumulated across installs 1..N. The defensible asset is the *slope*, not the day-one size, and install #1 already gets the full local tier-0 gate (zero-egress, standalone) on day one. The network adds resolution only on the tier-0 INCONCLUSIVE minority, and only once an operator activates the seam. The gate earns the install; the network keeps it.

> ⬡ **Enterprise-readiness matrix: built / flag-on / buyer-provisioned. Built and on by default:** the local tier-0 gate, all postures, the SDK, the stdio proxy with persistent pins (the gateway pin is per-process today, §4.2), the local telemetry/error sink (local capture on; egress plane held/off by default), the in-process tenant-isolation primitives. **Flag-gated (ships off; single-tenant is the launch default):** multi-tenant mode (`MCPINDEX_GATEWAY_MULTI_TENANT`). **Buyer-provisioned (held seams, §10):** the cloud tier-1 corpus, the tier-2 LLM consult, real OIDC/JWKS tenant identity, the KMS-backed signer, the durably-chained SIEM tenant log, and multi-region HA.

7. The dogfood proof

Here is the gate catching the silent change, reproducibly. Behind the shipping mcpindex proxy, a tool silently added a `required` parameter (`owner`) mid-session. The gate caught it and **held the call before the agent ran it** — a JSON-RPC `-32010` the tool never saw, never forwarded upstream. That is the whole product in one event: a contract moved after it was trusted, and the call stopped at the gate.

The systematic battery makes the claim reproducible. A deterministic dogfood server (`scenario_server.py`) mutates a single tool (`make_report(title, count[0..1000], mode∈{fast,full})`) along one safety-relevant dimension per scenario, driven through the **real proxy**. `verify_scenarios.py` asserts the gate's *actual* verdict matches the prediction across its `CASES` list and **prints its literal tail `14/14 scenarios PASS`** (verified by re-running `uv run --with mcp python verify_scenarios.py`). The cases cover the breaking, behavioral-mandated, additive, cosmetic, marker, and unknown-tool legs. (The verifier asserts 14; the table below displays 15 — it adds the byte-identical `base` baseline as a display-only reference row. The full row-count reconciliation lives once in the Appendix.) "Pass" means the gate's verdict matched the prediction, **including the cases where the correct verdict is INCONCLUSIVE and Guard holds the call.** INCONCLUSIVE is *not* a softer PROCEED.

SCENARIO	CHANGEKIND	GUARD VERDICT	WHAT IT PROVES
`base`	(baseline)	—	the pinned baseline (no verdict; the byte-identical reference)
`benign_noop`	(none)	PROCEED	unchanged contract → "matches your pin"
`added_optional`	added-optional-param	PROCEED	structurally-benign allowlist: auto-accept + re-pin
`added_required`	added-required-param	HOLD	a new mandatory parameter (`owner`): the hero shape
`removed_param`	removed-param	HOLD	a relied-on param vanished
`type_changed`	type-changed	HOLD	`count` int → string
`enum_reduced`	enum-values-removed	HOLD	`mode` [fast, full] → [fast]
`constraint_narrowed`	constraint-narrowed	HOLD	`count` max 1000 → 10 + **`additionalProperties:false`** (compound row; HOLDs on either)
`annotation_flip`	annotation-flip-to-destructive	**INCONCLUSIVE**	behavioral-mandated: tool now declares itself destructive → run-behavioral auto-offered; **Guard still holds**
`output_added`	output-schema-added	PROCEED	additive first-time `outputSchema` (still marker-scanned)
`output_changed`†	output-schema-changed	**INCONCLUSIVE**	behavioral-mandated parser-break (A→B); **Guard holds it**
`description_change`	description-only	HOLD	description-poisoning gate
`new_tool`	added-required-param‡	HOLD	brand-new un-pinned `danger_delete` → fail-closed; the HOLD is the unknown-tool / fail-closed leg, not a classified drift
— wire-fidelity, not recall (marker leg; not counted as independent structural detections) —			
`marker_input`	(—; carrier added-optional-param)*	HOLD	injection/exfil marker in an input-schema field
`marker_output`	(—; carrier output-schema-added)*	HOLD	exfil marker in a first-time `outputSchema`

The marker scan's honest job. The two marker rows are a **zero-cost early-warning that fires even on a byte-identical contract** — the one case where the structural diff is silent — and are explicitly **not** counted

toward any detection-rate claim: they prove the marker leg *wires* a HOLD, not that the scanner defeats an obfuscated marker (it does not, §4.3). They are visually separated above so the `14/14` pass-count is not read as 14 independent structural detections.

The three honest caveats, in one place:

> ⬡ **What this battery is and is not.** (1) The within-session hero rig (`verify_live_drift.py`) drives the HOLD through a **test client over the same MCP stdio transport and `list_changed` re-list contract a compliant host uses**; it is *not* a recording inside any host app, and no real-world public MCP server has been gated through the HTTP path yet (§8.2). (2) The `marker_input` / `marker_output` rows prove **wire-fidelity for a cooperative marker**, not adversarial recall: they show the marker leg drives the HOLD, not that the scanner defeats an obfuscated marker (it does not, §4.3); the `marker_output` leg is independent — it is scanned first and fires even on a hash match (§4.3 "Second"). (3) **The hero rig runs through the shipping proxy on its real persistent pin path.** `verify_live_drift.py` launches the proxy with no `--pin-store` argument, so the proxy uses its default `~/.mcpindex/pins/` store — the *same* persistent path the product ships. The within-session HOLD works because the base pin is set on the first `list_tools` , the drift fires mid-session, and the original baseline survives the re-list (the re-list is a TOFU no-op), so the next call HOLDs. What the rig does **not** yet exercise is the across-**restart** catch (kill the proxy, drift the server while it is down, relaunch, assert the held `-32010` on the first post-restart call); that runnable rig is §10.

The behavioral-tier rows (`annotation_flip` , `output_changed`) prove **the route, not an observed battery**: the dogfood asserts that Guard routes these kinds to INCONCLUSIVE and HOLDs, but the behavioral battery itself is not run in the rig (no isolator is wired), so what is proven is the route to INCONCLUSIVE and the Guard HOLD, not an observed CLEARED/FAILED. * The expected-kind on a marker row is `—` (the live verifier asserts no structural kind); the parenthetical names the structural *carrier* the diff happens to classify, but the HOLD is produced by the gate's **marker** leg. ‡ `new_tool` 's HOLD is the unknown-tool / fail-closed leg; the live verifier classifies its required params as `added-required-param` against the empty baseline, the structural carrier of the same HOLD. † `output_changed` is the one scenario that does not mutate the single global `base` : it pins a per-scenario baseline already carrying `outputSchema` "schema A," because an output-schema *change* presupposes a pre-existing output schema you cannot mutate from absence.

---

8. Deployment & data flow (and the security & procurement review band)

One question, the same shared gate, three forms. The pin store, the gate decision, and the fail-closed logic are extracted once (`cse.gate.Gate`) and reused verbatim across every surface, so there is exactly one decision path and no second copy to drift out of sync. Posture (default Guard) is operator-selectable per install.

8.1 The trust boundary

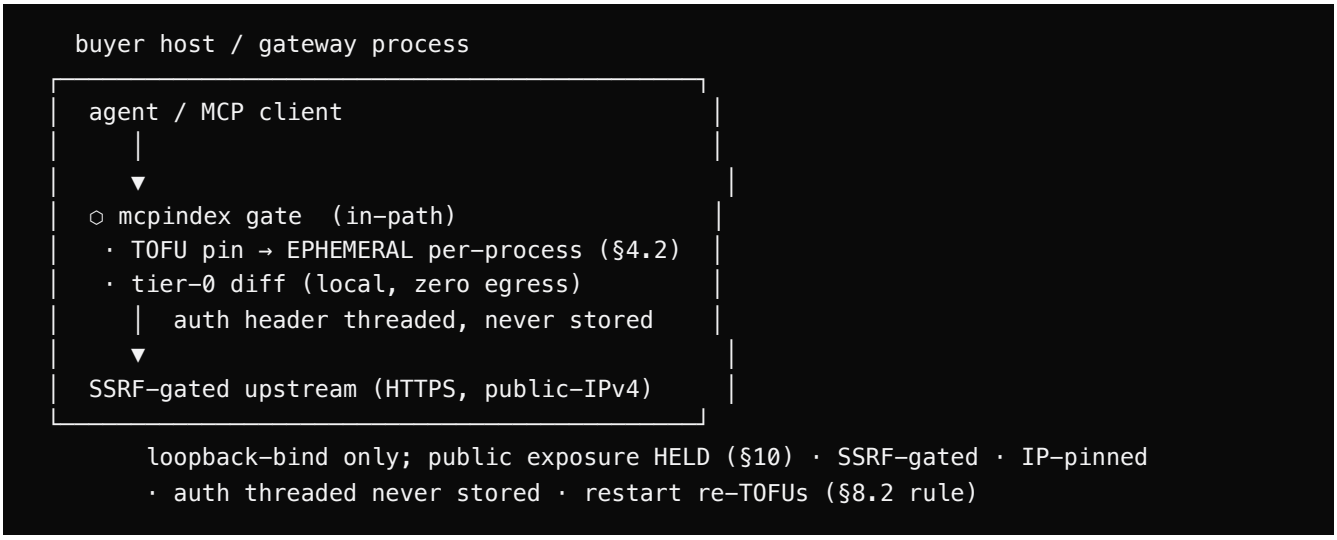
> ⬡ **The deal-closing headline.** The gate runs entirely on your host. The *only* thing that can leave is a SHA over the public contract (the `definition_hash`), and that egress is **off by default**.

The two deployed surfaces have **distinct** trust boundaries; they are drawn separately because they are not the same picture. **The HTTP gateway is the least-mature surface (§9 #9): loopback-only, fixture-**

tested, ephemeral per-process pin — form your impression of it from §8.2/§9 #9, not the boundary figure alone.

(1) The stdio proxy / SDK boundary (persistent pins, host-authenticated upstream):

(2) The HTTP gateway boundary (ephemeral per-process pin, transit-only auth, remote upstream):



Shared egress (right of both boundaries, off by default):

So a buyer reads the *proxy* boundary as persistent-pin + subprocess upstream (no SSRF, no auth-thread), and the *gateway* boundary as ephemeral-pin + loopback-bind + SSRF-gated/auth-threaded HTTPS upstream — never the two fused. (Render note: both boundary figures carry the ⬡ glyph on the gate node and amber on the `definition_hash ONLY` egress label only; the most-screenshotted artifacts, branded.)

What crosses the wire:

DATA	DESTINATION	DEFAULT	NOTES
`definition_hash` (one-way SHA-256 over public contract)	held cloud seam (tier-1)	**OFF**	the only persistent-store egress; opt-in, endpoint-pinned. **The precise property:** the hash is **one-way / preimage-resistant**, so a *private/internal* tool's plaintext never leaves (the confidentiality claim is "the hash is one-way," NOT "the contract is public"). It is **not unlinkability of a known public contract**: for a contract an adversary already knows, the hash is *confirmable by enumeration* against the corpus — a membership signal, not a secret. The cloud client runs the full `_resolve_and_check` allowlist on the operator-supplied URL too, but since the operator chooses that URL the control is **rebind/metadata-address prevention, not classic SSRF** (§2.5)
`{definition_hash, verdict, reason_class}`	contribute path (tier-1)	**OFF**	anonymized triple; closed-vocab `reason_class`
bounded contract-diff prompt (description + schema excerpt, ≤4096 chars)	tier-2 LLM endpoint	**OFF**	default `HeldLLMJudge` abstains, zero egress; on operator-provisioned model only, to the operator-configured endpoint
anonymized outcome aggregate / scrubbed error breadcrumb	telemetry / error sink	**OFF**	local capture is always-on, zero-egress; the egress plane defaults to `HeldEmitter` (no-op); `DO_NOT_TRACK`, enterprise tier, or one-click-off hard-veto egress
tool args / schema text / tool name / credential / request + response body	—	**NEVER**	never sent to any mcpindex sink, never logged, never persisted (a body transits only to the legitimate authenticated upstream, the proxy's function, §4.6)

The at-rest store meets the wire bar. The local telemetry/outcomes store carries **no request/response bodies, no arguments, no credentials**: closed-vocabulary event classes plus the public `definition_hash` only, the same bar §8.1's egress table meets for the wire. An over-cap or unparseable body fails closed to a breadcrumb that records **only the error class and the observed byte length, never any body bytes, even truncated.**

What a single-tenant buyer gets at launch. The single-tenant launch default (what §8.2 says a buyer deploys) gets a **computed on-device 30-day local stats card** (`cse/stats.py` `LocalStats`, fed by *both* the proxy and the SDK via `gate_stats.record_gate_call`, zero-egress): calls checked, HOLDs, re-pins/overrides, tier escalations, added latency, month-over-month catches, plus an on-host JSONL error/telemetry sink and an on-device outcomes store. So "how do I monitor gate HOLDs — and how noisy is Guard on my traffic" is answered by *a computed local card*, not raw log-grepping. There is **no mcpindex-operated fleet aggregation, no SIEM push** at launch; the decisions land in a local structured log each host's own agent can forward. A durable, exportable, multi-tenant decision feed a SOC ingests directly is the enterprise §10 SIEM seam, not the launch default.

Data residency, retention, sub-processors. All stores are on-host today: pins (`~/.mcpindex/pins/`), the local verdict cache, the per-tenant audit ledger, and the local telemetry/error capture (`~/Library/Logs/mcpindex-errors.jsonl`). Pins persist until re-pinned; the audit ledger and the local error log are bounded (ring-buffer / size-capped rotation); retention is configurable by the operator. **No buyer data reaches any sub-processor in the default (held) configuration.** When an operator enables the *canonical mcpindex* cloud seam, tier-1 runs on **Vercel + Supabase (US-region at launch; EU-residency is a §10 item)** as named sub-processors, provisioned in lockstep with seam activation (§5.2);

the only data that transits is the `definition_hash`, and a DPA / sub-processor disclosure attaches at enable time.

Cloud-corpus at-rest posture (the Network/Enterprise tier the buyer pays for — distinct from the on-host "N/A"). The on-host "encryption at rest: N/A" answer holds only for the default configuration, where stores carry no secrets. The *moment* the cloud seam is enabled, `definition_hash` values land in Supabase: **Supabase-managed AES-256 at rest**, and the corpus holds **only one-way `definition_hash`es plus closed-vocabulary verdict/reason-class data — no arguments, no schema text, no credentials, no tenant-segregated PII**. Whether corpus rows are tenant-attributable, the key-custody model for the cloud backend, and any tenant-segregation guarantee are **specified in the DPA at enable time** — the on-host "N/A" does not stand in for the cloud case, and a buyer evaluating the paid tier reads this row, not the on-host one.

Compliance / data-handling posture. In the default configuration nothing leaves the host, so there is no customer or personal data egress. When the cloud seam is enabled, the only datum that transits is a one-way SHA over a *public-contract field-set*, not personal data, not customer content, which keeps the GDPR/CCPA surface minimal; the corresponding DPA attaches at enable time.

For an EU buyer specifically. At launch the default posture is **on-host, zero cross-border transfer** — no datum leaves the EU host because no datum leaves the host at all. The only state in which anything transits is an operator-enabled cloud seam, and then the single datum is the one-way `definition_hash` to a **US-region** sub-processor set (Vercel + Supabase), governed by **Standard Contractual Clauses** in the DPA that attaches at enable time; the contract over which the hash is computed never leaves. **EU data residency for the cloud seam is a named §10 item** — until it lands, an EU buyer with a hard in-region requirement should run on-host only (the launch default), which transfers nothing.

8.2 The surfaces

- **Host plugin / one-click install.** Two install paths ship and execute as written:

```
curl --proto '=https' --tlsv1.2 -fsSL https://mcpindex.ai/install.sh | sh
#   --dry-run  show the wiring plan, write/register nothing
#   --yes      skip the per-host wiring confirmation
# or:  double-click the Claude Desktop .mcpb bundle (the host-plugin path)
```

  One install fronts the host's MCP servers through a single local gate process; posture and pins are per-host.

  > ⬡ **What `install.sh` actually does, disclosed, not hidden.** The script (1) installs the local proxy; (2) auto-detects every MCP host and, **with your affirmative consent** (it echoes the plan and asks, unless you pass `--yes`), rewrites each detected host config to route through the proxy — each rewrite atomic, marked for in-place un-wire, byte-restorable; and (3) registers a resident **login-item watcher** (a **user-level** launchd LaunchAgent on macOS / systemd *user* unit on Linux, **not root**) that auto-wires servers added later. Everything is idempotent and reversible: `uninstall.sh` un-wires every host and removes the watcher. `--dry-run` shows the full plan before anything is written. We disclose the daemon and the config rewrites because "audit exactly what runs on your host" (§8.3) is the promise. The flags the installer accepts are exactly `--yes` / `--dry-run` / `--help`; it rejects anything else.

> ⬡ **Locked-down / no-rewrite install path (the manual variant).** A buyer whose endpoint-security policy forbids config rewrites or resident daemons can install **proxy-only, with no host-config rewrite and no background watcher** — *without* running `install.sh` at all:
>
> ```
> uv tool install mcpindex-preflight          # installs the proxy artifact only — no r
> python -m tooling.cse.config_wire wire --config <your-host-config-path>   # wire ONE
> # unwire later:  python -m tooling.cse.config_wire unwire --config <path>
> ```
>
> You trade auto-wiring for zero config-rewrite-by-the-installer and zero background daemon; the gate still holds drift on the wired client, and `config_wire` writes the same atomic, byte-restorable, marked-for-unwire entry the installer would. This is the install variant most likely to clear an endpoint-security gate. *(There is no `install.sh --proxy-only` flag; the manual `uv tool install` + `config_wire` path above is the verified, shipping way to get a no-rewrite, no-watcher install.)*

> ⬡ **Endpoint-security posture (macOS notarization + EDR).** For a managed fleet: the resident config-rewriting watcher (the `install.sh` path) may trip EDR behavioral detections on config writes and is the case most likely to need an EDR allow-list entry — so the **manual no-rewrite / no-watcher path above is the EDR-friendly default for managed endpoints.** Apple notarization / code-signing of the install artifacts at launch is disclosed in the security band (§8.3); SLSA/sigstore build provenance is §10. (The published package names under one brand — `mcpindex-preflight` (PyPI installer) / `mcpindex-trust` (dev source tree) / `mcpindex` (Claude Desktop bundle manifest) / `@mcp-index/sdk` (the scoped npm TS SDK) — and the `tooling.cse.*` module path a buyer audits are itemized in the Appendix package-identity note.)

- **SDK wrapper.** One line around a session the host already authenticated — **the `wrap()` library, not a standalone CLI; you call it around your existing session.** The published TS package is the scoped **`@mcp-index/sdk`** (PolyForm-NC); it is scoped deliberately because the bare `mcpindex` npm name is similarity-blocked against the unrelated third-party `mcp-index`, so `npm install mcpindex` would not resolve to us:

  ```
  // TypeScript — the shipped signature is wrap(session, opts); `pin` is REQUIRED.
  import { wrap, PreflightPin } from '@mcp-index/sdk';   // npm install @mcp-index/sdk
  // Pass an empty PreflightPin store to TOFU-pin on the first list_tools:
  const wrapped = wrap(session, { pin: new PreflightPin(), serverId });
  ```

  Python today is the proxy/CLI path, not a library `wrap()`. The Python `wrap()` exists in the engine but only at the internal `tooling.cse.preflight_intercept` module path — there is **no published, `pip install`-able Python SDK** that exposes it at launch (a standalone PyPI Python-SDK import surface is a §10 item). A Python user gets the same coverage today through the genuinely-shipping **`mcpindex-preflight`** proxy/CLI (`pin` / `check`, §8.6) and the stdio proxy; we name this rather than show a Python `import` line that does not resolve.

  A held call surfaces to the integrator as a **structured held verdict** (mirroring the `-32010` contract); you catch it and route the agent to Review / Re-pin / Validate. The TS gate is a faithful port of the Python gate, down to the ⬡ banner.

> ⬡ **For the integrator (the day-one developer path).** The held-verdict object an SDK caller catches carries the same closed-vocabulary fields as the wire `-32010`: the subject (`server_id`, `tool_name`, hash old→new), the `ChangeKind`(s), the verdict, `tier_reached`, `verdict_scope`, and the bounded before/after evidence — **never arguments, never a credential, never a body** (§5.4). The `wrap(session, { pin, serverId })` config surface is: the **required** `pin` (a `PreflightPin` store — pass `new PreflightPin()` empty to TOFU-pin every tool on the first `list_tools`, or a populated store to diff against an existing baseline; `WrapOptions.pin` is a required field, not optional), `serverId` (the pin-store key), and the per-install posture (Monitor / Guard / Strict, default Guard, §4.4). The Trust API is rate-limited per the Directory API (Pro) tier (§6) and authenticated by an API key on the Pro path; the unauthenticated path serves the public four-state read. Full request/response shapes and the SDK reference are at **/docs** (published in lockstep with go-live).

- **Trust API (the directory screen).** A real deployed route returns the four-state decision with its evidence. At launch, with the corpus still graduating (§5.6), the common live response is `UNVERIFIED`:

```
GET /api/v1/trust/tool/{server_id}/{tool_name}
// screened tool (the response contract shape):
→ {
    "subject": { "server_id": "uptimerobot", "tool_name": "get_monitors" },
    "status": "OK",
    "directive": "ALLOW",
    "granularity": "description",
    "dimensions": [ /* per-dimension findings */ ],
    "expires_at": "2026-07-01T00:00:00Z",
    "honest_limits": [ "advisory", "description_granularity" ],
    "verdict_contract_version": "1.0.0"
  }
// launch-state common case — no verdict on file (fail-closed; the agent must NOT infer t
→ {
    "subject": { "server_id": "...", "tool_name": "..." },
    "status": "ERROR",
    "directive": "UNVERIFIED",
    "granularity": null, "dimensions": [], "expires_at": null,
    "honest_limits": [ /* no-verdict limits */ ],
    "verdict_contract_version": "1.0.0"
  }
```

  Field bridge to §3: **`directive`** is the §3 four-state verdict (ALLOW/DENY/REVIEW/UNVERIFIED); **`granularity`** is the clearance-level (§3 `VerdictGrade`); **`honest_limits`** is the per-verdict limit set; UNVERIFIED is the status-only no-verdict state. A server-level lookup is at `GET /api/v1/trust/server/{server_id}`. A hash-keyed `verdict/{definition_hash}` lookup is planned, not live (§10). The browsable directory surfaces are at **/screen, /stats, /leaderboard, /status, /changelog** (§3).

- **MCP proxy.** A transparent in-path proxy on the JSON-RPC wire that HOLDs at the transport layer (`-32010` / `-32011` / `-32012`, §4.5), the form that gates a host with no SDK change. **Local stdio and `http://localhost` servers are covered by this proxy and the SDK wrapper.** Persists pins by default.

> ⬡ **Remote server, today, on the proxy.** If your MCP server is remote but you want persistent-pin coverage today, point the local stdio proxy at it rather than waiting for the gateway: run the proxy on the host that holds the upstream's credentials, wire your client through it, and the proxy gates the remote server with the persistent `~/.mcpindex/pins/` baseline — the across-restart catch the gateway does not yet have. The remote upstream is reached over the host's existing authenticated transport; **there is no SSRF gate on this proxy path** (only the HTTP gateway path is SSRF-gated). The gateway is the eventual multi-tenant front; the proxy is the high-assurance remote path now.

- **HTTP / gateway.** Gates *remote* servers by threading the host's existing auth through a pass-through (§4.6), adding only a front end plus the tier-1 escalation inside the shared core — no second gate, no new transport, **no request or response body logged**. The mechanism is a positive allowlist: the gateway gates upstreams to an `https://` URL that resolves to a **public IPv4**; plain-http, loopback/LAN, CGNAT / tailnet (RFC 6598, `100.64.0.0/10`), and **any IPv6 upstream are refused** by the SSRF gate today. The gateway **binds loopback by default** (`assert_gateway_loopback`); public exposure is held behind signed-binary and consent work (§10). Its pin is per-process today, with the restart-drift consequence stated in §4.2.

  > ⬡ **Decision rule for the gateway restart-drift hole.** A platform lead deploys, autoscales, and crashes routinely, so the ephemeral per-process pin re-TOFUs as *normal operations*, silently accepting a drifted contract. **A re-TOFU starves the whole ladder**: like the withheld-`list_changed` seam (§2), the corpus keys on a hash the gate only computes against a real baseline, so tier-1 coverage cannot rescue a gateway that re-pinned the drift; baseline integrity is upstream of every tier. **The launch operating rule: for any remote MCP server where drift-across-restart matters, use the single-tenant proxy path** (persistent pins today, recipe above) **or mount a durable pin path on the gateway**; do not rely on the default ephemeral-pin gateway for high-assurance remote servers until per-route persistent pins land (§10). Interim, in preference order: (a) prefer the proxy path; (b) mount a persistent pin path; or (c) run a startup re-validation step against a trusted baseline before serving.

  The only upstream with a test fixture today is `uptimerobot` (a **smoke fixture**, `smoke_gateway_auth.py`): it proves SSRF refusal, auth pass-through, and no-body-logging against a fixture, not a live gate against the real server; **no real-world public MCP server has been gated through the HTTP path yet.** A GitHub-style HTTPS+public-IP server is in-perimeter-but-unproven (coverable, not yet wired or tested, §9 #9). The genuinely dogfooded HOLD is the §7 stdio proof. **Server grade, stated once:** the launch gateway is a Python-stdlib threaded loopback server (`ThreadingHTTPServer` / `BaseHTTPRequestHandler`: daemon threads, per-request read timeout, ~2 MB body cap) — appropriate for a loopback single-tenant front; a production WSGI/ASGI front and concurrency tuning are part of the §10 "open the gateway" work.

- **Enterprise / cloud.** What is **built and unit-proven today**: the logical (in-process) multi-tenant **isolation primitives**, per-tenant route maps, verdict caches, rate budgets, and append-only event ledgers (`TenantAuditLog`, §6), with no shared mutable state and a fail-closed 404 on an unknown tenant (not VM/container isolation). A request maps to a tenant by the `/t/<tenant_id>/` path prefix (preferred) or the `X-Mcpindex-Tenant` header (**unsigned, spoofable today**); the `tenant_id` is validated like a route name and an unknown/malformed id fails closed (404/refuse). Multi-tenant mode ships **OFF by default** (`MCPINDEX_GATEWAY_MULTI_TENANT`); single-tenant is the launch default. A process compromise crosses the logical boundary, so a high-assurance tenant should expect per-tenant process/VM separation: roadmap, not today. Tenant *isolation* is only as strong as tenant *authentication*: spoof-resistant identity (OIDC/JWKS) is §10.

> ○ **The multi-tenant operating rule.** Single-tenant is **ready now** and is the launch default. **Do not enable `MCPINDEX_GATEWAY_MULTI_TENANT` for untrusted-tenant separation at launch:** a tenant is authenticated by an *unsigned, spoofable header/slug* into a *shared-process* (logical, not VM/container) boundary, suitable only for cooperative, same-trust-domain tenants. Hardened multi-tenant with spoof-resistant identity (OIDC/JWKS) and per-tenant process/VM separation is buyer-funded provisioning (§10).

Single-writer on the write plane; stateless on the read plane. Split by plane, because the two scale differently: the **read/serve plane is stateless and horizontally scalable by design** (`enterprise/serving.py` — identical stateless nodes behind the edge), so verdict reads N-node out. The **write plane — the append-only `DriftCorpus` chain plus the verdict/pin stores — is single-writer** (SQLite WAL): one host, one writer; the append-only chain cannot be fanned across concurrent writers. So enterprise HA is **active-passive failover on the *writer*, not multi-writer scale-out**: held decisions are fail-closed during a failover (no positive clearance minted while degraded), and SLA/RTO is buyer-negotiated (see §8.5 for the default target).

8.3 Security assurance

mcpindex is pre-SOC-2; we are honest about that and lead with the assurance the open-core model gives a buyer *today*. **The in-path client is open-source and import-firewalled (the open-core import firewall, `open_core_manifest.py`, §6 governance axis); a buyer can audit exactly what runs on their host**, and the §7 dogfood is reproducible evidence of what the gate does. **What independent assurance exists today, stated plainly:** no third-party pen test or SOC 2 at launch (§10). Available *now* for review: the open-source in-path client + the import firewall, the reproducible §7 dogfood, and (at launch) the published SBOM + `security.txt`. There is no third-party security assessment at launch; we say so rather than let a buyer discover it mid-review. A third-party penetration test and SOC 2 Type 2 are targeted when enterprise demand warrants (§10).

Control-framework posture (for a GRC gatekeeper). A **CAIQ-Lite / SIG-Lite self-assessment is available on request** at the design-partner stage, so a security reviewer is not left to build the crosswalk from prose. The architecture is **designed toward the SOC 2 TSC Security + Confidentiality criteria** (zero credential custody, closed-vocabulary stores, fail-closed boundaries) — designed-toward, not attested, until the §10 Type 2. On AI governance: the verdict is **advisory and human-in-the-loop by design** (a HOLD routes to a human; the gate mints no autonomous clearance, §5.5), which is the relevant posture for an EU AI Act / NIST AI RMF classification — mcpindex is a change-detector that defers the decision to a person, not an automated decision system. None of this substitutes for the §10 third-party attestation; it gives the gatekeeper the map.

The as-is / no-warranty risk-transfer asymmetry, named. At the design-partner stage the assurance model is **buyer-led code audit plus the reproducible §7 proof, not vendor warranty or third-party attestation**. A risk team should read this plainly: a buyer who needs contractual risk-transfer should treat SOC 2 + pen test (§10) and negotiated liability as part of the engagement, not assume the open-source posture substitutes for them. Naming the asymmetry builds more trust than implying the audit covers it.

Install integrity & supply chain. The install artifact is the PyPI package `mcpindex-preflight`, which is **hash-pinnable** (pin the version + hash in your own lockfile/policy). We commit to **publishing a checksum/signature for each release** alongside `security.txt`; SLSA/sigstore build provenance is a named §10 item, not implied as present. On macOS, the install artifacts are **notarized / code-signed at launch** (so Gatekeeper does not block them); the resident config-rewriting watcher may need an **EDR**

allow-list entry on a managed fleet, and the manual no-rewrite/no-watcher path (§8.2) is the EDR-friendly default for that case. The resident watcher runs at **user level** (LaunchAgent / systemd user unit, **not root**, §8.2). For a locked-down endpoint, the **manual no-rewrite install path** (§8.2: `uv tool install mcpindex-preflight` + `config_wire`, no `install.sh`) installs the proxy with **no host-config rewrite and no background watcher** — the install variant most likely to clear an endpoint-security gate. The honest substitute for a not-yet-SOC-2 state is auditability: read the code that runs on your host, pin the artifact, and reproduce the proof.

For a downstream security review today: a small-team buyer who is themselves asked "is your vendor SOC-2?" can hand a non-technical procurement gatekeeper something forwardable: the in-path client is open-source and import-firewalled — point your reviewer at the repo and the reproducible §7 proof; a DPA + sub-processor template is available on request.

- Security & Procurement Quick-Reference (paste into your vendor questionnaire).

QUESTIONNAIRE FIELD	ANSWER AT LAUNCH
Data egress	`definition_hash` only (one-way SHA over a public-contract field-set); **off by default** (§8.1)
Data residency	on-host by default; US-region cloud at enable (Vercel+Supabase, provisioned in lockstep with seam activation); EU/multi-region §10
Sub-processors	none in default config; Vercel + Supabase when the cloud seam is enabled
Encryption in transit	HTTPS, IP-pinned upstream; endpoint-pinned cloud path. The IP-pin prevents rebind/metadata-address reach; on the operator-chosen cloud URL the control is rebind/metadata prevention, **not classic SSRF** (§2.5)
Control-framework posture	CAIQ-Lite / SIG-Lite self-assessment available on request; designed toward SOC 2 TSC Security + Confidentiality (not attested until §10)
AI-governance classification	verdict is advisory + human-in-the-loop by design (relevant to EU AI Act / NIST AI RMF); not an automated decision system (§5.5)
Vuln remediation SLA	target: critical patch + advisory within 5 business days, high within 15; design-partner-negotiated (§8.5)
RTO / RPO	read plane stateless/N-node; write plane single-writer active-passive; default best-effort, no contractual SLA at design-partner stage (§8.5)
Encryption at rest	**On-host default:** N/A — stores carry no secrets (public contract hashes + public schemas only, §8.4); inherit the host file-permission model; pin store not integrity-checked on load (§8.4 host-trust boundary). **Cloud seam (Network/Enterprise tier):** Supabase-managed AES-256; corpus holds only one-way `definition_hash` es + closed-vocab verdict data, no PII; tenant-attributability + key custody specified in the DPA at enable time (§8.1)
Credential storage	**none**: SDK custody-free, gateway transit-only, never stored/logged (§4.6); see the in-path blast-radius callout (§4.6)
Request/response body	transits to authenticated upstream only; never logged/persisted/sent to mcpindex
At-rest telemetry store	closed-vocab + `definition_hash` only; no bodies/args/credentials (§8.1)
Install integrity	`mcpindex-preflight` PyPI, hash-pinnable; checksum/signature at launch; SLSA §10
Watcher privilege	user-level (LaunchAgent / systemd user), not root; manual no-rewrite variant available (§8.2)
Multi-tenant authn	header/slug-trust today (cooperative tenants only, fail-closed on unknown id); OIDC/JWKS §10
Audit / decision export	local on-host JSONL + computed stats card today; durable SIEM export §10 (prerequisite, not enhancement)
Incident response / breach notify	§8.5: vendor-side compromise notification (72h target) **with named detection & containment** per vector (cloud corpus: manual contribute-anomaly review + seam-disable; supply chain: checksum-mismatch monitoring + yank/advisory), distinct from the (minimal) buyer-data-breach surface
Independent assurance	none third-party at launch; open-source client + import firewall + reproducible §7 proof + SBOM/security.txt today; pen test + SOC 2 §10
SOC 2	pre-SOC-2; Type 2 roadmap §10
CVD / disclosure	`security.txt` at launch; target first-response 2 business days

QUESTIONNAIRE FIELD	ANSWER AT LAUNCH
Availability / RTO	read plane stateless/horizontally-scalable; write plane single-writer active-passive failover; RTO buyer-negotiated (§8.5)
Nature of a verdict	advisory contract-diff, **not a warranty of safety** (§9 #1)

Contractual posture (design-partner stage). The in-path client is PolyForm-Noncommercial and provided **as-is, no warranty**, which is *why* the buyer's operating posture (route every HOLD to review; treat the gate as a change-tripwire) is part of the deal, and why durable SIEM export and KMS-signed receipts are negotiated, not assumed. mcpindex does not warrant that a PROCEED is safe or that a HOLD fired on every drift. A **DPA + sub-processor-list template is available on request during evaluation**. Limitation-of-liability and indemnity are negotiated per engagement at the design-partner stage. The §9 limits are the contract.

8.4 mcpindex's own attack surface

The gate is a new in-path component on every tool call, so the first question is the blast radius when the gate *itself* is the target. (The credential-transit blast-radius answer is consolidated at §4.6; this subsection covers availability, parser/DoS, and the pin store at rest.)

- **Host-level availability / fail direction.** The proxy wraps every `gate.decide` in a graceful-fail guard: a bug or unexpected exception in mcpindex's own code **never crashes the relay or hangs the pipe**; it fail-closes to an internal-error HOLD (`-32012`) plus a scrubbed breadcrumb. The host-level fail direction is **closed**, and that internal-error HOLD is always fail-closed regardless of the `fail_open` posture flag. Monitor is the explicit break-glass during an outage (§4.4).
- **Parser / DoS resistance.** The structural diff is bounded by `MAX_DEPTH = 16`; every gateway/proxy body is bounded to a ~2 MB frame cap (over-cap or unparseable fails closed); and the tier-2 prompt is truncated to 4096 chars so a pathological diff cannot inflate a metered model call.
- **Pin-store tamper model.** The pin store is plain on-host JSON under the user's home (`~/.mcpindex/pins/`), written atomically (mkstemp + `os.replace`). It inherits the host's file-permission model: a local process running **with the user's own privileges** can read or rewrite it, but such a process is already inside the §2 "compromised host" boundary. **The pin is not integrity-checked on load: a same-privilege local process that rewrites it is trusted on next load**, consistent with the §2 host-trust boundary; pin-store signing is a candidate hardening, not a launch feature. The store carries no secrets (only public contract hashes and captured public schemas) and is not a privilege-escalation path. The pin baseline is as trustworthy as the user account that owns it.
- **Input-validation boundary.** Every attacker-derived field reaching a user-visible surface is control-byte-stripped and green-word-redacted (§2.5, §4.5); every cloud-tier-1 response is closed-vocabulary-validated and endpoint-pinned; the tier-2 description is treated as hostile input and cannot mint a clearance (§5.3).
- **Dependency / SBOM posture.** The in-path client's defensible property is its small, auditable surface: stdlib plus a tight set of first-party modules and the MCP SDK, enforced by the open-core import firewall (`open_core_manifest.py`, §6 governance axis). We commit to publishing the in-path client's SBOM in a **standard machine-readable format (CycloneDX, with an SPDX variant on request)** alongside `security.txt` at launch — so a post-xz reviewer can ingest it directly and score the **single-digit-to-low-tens transitive-dependency count** rather than take "small auditable surface" on faith.

8.5 Procurement & operational posture, incident response & breach notification

- **Operational SLAs (design-partner stage, stated not silent — with the path to contractual). Vuln remediation:** target a critical patch + advisory within **5 business days**, a high within **15**, on the CVD / `security.txt` channel; design-partner-negotiated, not yet a contractual SLA. **Availability:** the read/serve plane is stateless and horizontally scalable; the write plane is single-writer active-passive (§8.2), so the default posture is **best-effort RTO/RPO, no contractual SLA at the design-partner stage** — usable because it is stated, not because it is strong. **Path to contractual:** these design-partner targets become contractual SLAs scoped into the enterprise provisioning engagement (§10), at which point RTO/RPO and remediation windows are committed in the order, not best-effort. **Cloud-seam operational ownership:** when the canonical cloud tier-1 seam is enabled, mcpindex operates it on Vercel + Supabase (§5.2) with on-call and a **bearer-key rotation cadence committed in the DPA at enable time**; until the seam is activated there is no cloud operational surface to own.
- **Verdict-contract versioning.** The `verdict_contract_version` (`1.0.0`) is semver; a breaking change is a new major at a new path, announced with an RFC 8594 `Sunset` header on the deprecated route, so an integrator pinned to a contract version is never silently broken.
- **Buyer-data breach surface (minimal).** In the default configuration nothing leaves the host, so there is no buyer-data breach surface. When the cloud seam is enabled, the only datum that transits is the `definition_hash`; a breach-notification commitment for that seam attaches in the DPA at enable time.
- **Vendor-side compromise notification (distinct from the above).** The buyer-data surface is minimal, but mcpindex-the-*vendor* can itself be compromised, which is a separate obligation:
  - **Canonical cloud corpus compromise (Vercel + Supabase).** *Notification:* if the canonical cloud tier-1 corpus is breached, we commit to a vendor-side compromise notification within a stated window (target **72 hours**, design-partner-negotiated at the engagement). Because the only datum the corpus holds is the one-way `definition_hash`, the buyer-data exposure of such a breach is bounded by the non-invertibility of that hash (§8.1) — but the notification obligation stands regardless. *Detection & containment (stated honestly at the design-partner stage):* the detection control is **anomalous-contribute monitoring** — at the design-partner stage this is **manual review of contribute volume and reason-class distribution**, not automated anomaly detection (which is a §10 maturity item); containment is **disable the seam + invalidate the affected `definition_hash` rows + advisory via `security.txt`**. We name the immaturity rather than imply a detection capability the notification SLA would otherwise presuppose.
  - **Supply-chain compromise of `mcpindex-preflight`** (the in-path install artifact through which buyer credentials transit). *Detection:* **signed-release verification + checksum-mismatch monitoring** — a buyer who pins the version+hash (§8.3) is not silently upgraded onto a bad release, and a checksum mismatch against the published signature is the trip-wire. *Containment:* **yank the bad release + advisory via the CVD / `security.txt` channel** within the same target window. Hash-pinning is a *prevention* control; the detection-and-response capability is the checksum-mismatch trip-wire plus the yank/advisory path, stated so the notification SLA is backed by a detection story rather than only prevention. SLSA/sigstore build provenance (§10) strengthens detection further.
  - We explicitly distinguish the **buyer-data breach surface** (minimal, by design) from the **vendor-compromise notification obligation** (stated above) so a security reviewer is not left to infer the latter from the former.

8.6 Operating the gate in production (the day-2 model)

The platform is described above; here is the operating model in one place, so day-2 ownership is not assembled from four sections.

> ⬡ **The first 10 minutes after install — see it working, before any drift fires.** The modal buyer installs to protect tools they already use, so on day one, against existing servers, the gate is *watching* and the screen is *UNVERIFIED* — there is no banner yet. That silence is the gate working, but it is not legible as working unless you make it so. Three observable signals, in order:
>
> 1. **Fire a HOLD on your own machine in under a minute** — capture a baseline with `mcpindex-preflight pin`, hand-edit one tool's contract to add a `required` parameter, then `mcpindex-preflight check`: the gate HOLDs the drift and prints the ⬡ verdict on your host, so you *see* a HOLD fire and clear with a re-pin, not just read that it would. (For the full deterministic battery — the 14-scenario `verify_scenarios.py` rig behind the §7 proof — run `uv run --with mcp python verify_scenarios.py` from the dogfood repo; it is the reproducible HOLD a technical reader exercises end-to-end.)
> 2. **Read the pinned inventory** — the local stats card (§8.1) shows "N tools pinned, baselines captured," so "it's working" is visible at **zero drift**: every tool you use is now pinned and watched.
> 3. **Re-frame the steady state** — from here, no banner means *protected and quiet*. The first real banner fires the first time a vendor's contract moves, which may be days or weeks out; the pinned inventory + the HOLD you fired yourself with `mcpindex-preflight` are how you know the gate is live in the meantime.

- **Recommended rollout.** Start in **Monitor** → measure your own noise floor on the local stats card for a week → switch to **Guard** (§4.4). Strict is for the narrow set of servers where any safety-relevant drift must hold.
- **Where HOLDs surface.** In-client ⬡ banner at the moment of the call (§4.5) · the per-server daily digest (`cse/digest.py`, safety-relevant only, empty = healthy) · the computed local stats card (§8.1).
- **Who owns triage.** A HOLD routes to whoever owns the agent's runtime; recovery is one click (Review / Re-pin / Validate, §4.5).
- **Steady-state expectation.** Most vendor releases pass silently (structurally-benign auto-accept, §4.4); an empty digest is the healthy state, not a missing-data state. The taxonomy (§4.4) gives the *shape* of what holds; size your exact friction on your own traffic before you tune posture.
- **Audit-retention disqualifier, in one place. If you carry a regulatory log-retention or tamper-evident-audit obligation, treat durable SIEM export (§10) as a gating prerequisite, not an enhancement: the launch tenant ledger is an in-memory ring buffer (evicts oldest-first, not durably hash-chained) — operational telemetry, not a retained audit record.** A regulated buyer is not production-ready on the audit axis until the §10 SIEM export lands; we state this here rather than let it surface mid-procurement.

---

9. Honest limits (stated as strength)

A trust product that over-claims its maturity loses the trust it is selling. **In practice the gate catches the change that actually bites you**: a silently-added required param, a destructive flip, a narrowed constraint, a reduced enum, a removed param, a rewritten description, after your agent already trusts the

tool. The list below is the honest perimeter around that core, clustered into three groups. Each is tagged **intrinsic** (true at any version) or **at-v1** (maturity-gated, not a deploy flip). Each at-v1 limit names its §10 unblock so a buyer can trace "this blocks me today → here is what unblocks it."

> ⬡ **Read the list as confidence, not as a hedge.** Every limit below is one a competitor has and does not disclose. We disclose ours because a trust layer that hides its perimeter is not one. The §6 enterprise-readiness matrix maps built / flag-on / buyer-provisioned in one place.

A — What the verdict is (intrinsic).

1. **A verdict is a contract-diff, not a safety oracle.** A HOLD means *the contract changed versus your pin*, not *the new contract is unsafe*; it pauses the call for a human, and the human decides. *(This is the one load-bearing limit; its single canonical home is here.)*
2. **TOFU cannot catch pre-pin drift, and the gate is a *change* detector, not a malice detector.** A contract hostile from first sight that never moves produces no drift, the directory screen's domain (§3).
3. **The marker scan is a trip-wire for known, unobfuscated patterns, not an injection defense.** An attacker who controls the bytes encodes around it; a benign description with a marker-like substring can false-positive into a HOLD. Both directions cost a one-click Re-pin, never a silent miss. The real claim is the byte-diff (§4.3). A noisy marker scan feeds the adversarial HOLD-fatigue vector (Appendix #5), which is exactly why the recommended posture for marker-heavy vendors is **Monitor + the daily digest, not per-call Guard holds** (§4.4).
4. **Response-content injection is out of scope.** A real-world MCP vector lands in the data a tool returns at call time, on a byte-stable contract; the gate does not inspect the response stream (§2). We defend the contract because it is the surface that can be enforced in-path.

B — What is not yet calibrated (at-v1).

5. **`calibrated=false` until calibrated.** Confidence is in shadow mode behind `AUTONOMY_ENABLED = False`. The live outcome stream calibrates the held direction but cannot certify the cleared direction from "no complaint" (censored data, §5.5). We publish no recall/FP number yet; the D3 graduation gate stands at 15/150. **Unblock:** §10 "Calibrate" (build the held-out adversarial corpus).
6. **The directory screen is advisory and semantic; the conformance probe is monitored, not enforced.** The screen reasons about a published contract and is a *graduation milestone*. The in-path gate enforces the HOLD; the screen is the prior. **Unblock:** §10 "Graduate the directory" + "Enforce the behavioral tier."
7. **The behavioral tier clears or refutes; it does not guarantee safety.** A tier-3 CLEARED is dev-local and ephemeral; enforced positive clearance is a separate, gated one-way door. "Caught," "held," "cleared", never "blocks attacks," "tamper-proof," or "guaranteed safe." **Unblock:** §10 "Enforce the behavioral tier."

C — What is built but buyer-provisioned (at-v1).

8. **The tier-1 cloud corpus and the tier-2 LLM consult are held by default; the network is empty at launch.** Both seams are built and fail-closed (`HeldCloudTier1Client`, `HeldLLMJudge`, zero egress). The corpus *mechanism* is live; the corpus *data* is empty until an operator activates the seam (§5.2). A hash-keyed corpus verdict transfers a contract-judgment, not a behavior-level safety clearance. **The destructive subset of the corpus-poison vector is already closed in code today** — the shared `_tier1_adapter` M1 guard refuses a corpus `cleared` for any behavioral-mandated kind (§5.2). What

is still owed at §10 is the *general* re-derive-locally evidence standard for any future purely-structural INCONCLUSIVE class, which must land in the same change as activation; the residual is empty today by the §5.3 routing invariant (the canonical analysis is §5.2). Install #1's value is the local tier-0 gate. **Unblock:** §10 "Activate the tier-1 cloud network" (with the evidence-standard asymmetry in the same change).

9. **The remote/gateway path is the least mature surface.** No real-world public MCP server has been gated through the HTTP path yet (proven against a fixture only), and the gateway pin is per-process: the restart-drift hole is real until per-route persistent pins land (§4.2, §8.2 rule). **The composed worst case (named in §2):** on the ephemeral-pin gateway the withheld-`list_changed` seam and restart-re-TOFU *compose* — an adversary who induces a restart coincident with a withheld-notification mutation defeats both the in-session and the next-launch floor; use the persistent-pin stdio proxy for any remote server where this matters. **Coverage limit, not only security:** IPv6-only/dual-stack or Tailscale (CGNAT) upstreams cannot be gated through the HTTP gateway today; use the stdio proxy/SDK path (§8.2 recipe). **Unblock:** §10 "Open the gateway."
10. **Enterprise SSO, KMS-signed provenance, durable audit chaining, tenant authentication, and multi-region HA are buyer-funded provisioning, not deploy flips.** What is built is the fail-closed isolation primitives and the auth/signing *design*; what a buyer funds is the real adapters. **Unblock:** §10 "Enterprise provisioning."

None of these flip on deploy. Keeping them in the launch content is the difference between a trust layer and a marketing layer.

---

10. What's next (roadmap)

The §9 limits are not on this list as "coming soon"; they are intrinsic or genuinely unearned. What *is* on the roadmap is the maturity work and the buyer-funded provisioning, grouped by horizon, by the buyer-facing unblock each clears, and by the **trigger** that advances it (a trust buyer evaluates triggers, not dates — so we commit conditions, not dates).

Critical path, in one line. The across-restart proof and forced periodic re-list are the cheap near wins (no dependency); **tier-1 activation is gated on the evidence-standard-asymmetry change landing in the same commit**; and the autonomy flip is gated on the held-out adversarial corpus and depends on nothing else. Read the sequence, not just the grouped list.

Near — the install/monitoring loop hardens. *(Trigger: the install base generating drift signal.)*

- **Ship the runnable across-restart proof.** A `verify_restart_drift.py` rig (pin via the shipping proxy with `--pin-store`, SIGKILL, drift the server while down, relaunch, assert the held `-32010` on the first post-restart call), making the §4.2 persistence claim runnable, not only code-readable. *(The §7 within-session rig already runs the persistent path; this is the natural next leg.)*
- **Activate the tier-1 cloud network.** *(Trigger: a design-partner count / contribute-opt-in threshold that makes a populated corpus worth standing up.)* Stand up the operator-provisioned `CloudTier1Client`, turn on the contribute path, and let the corpus + flywheel compound. **Hard precondition (ordering gate): generalize the evidence-standard asymmetry at the shared resolution point in the SAME change that turns on contribute/lookup.** The behavioral-mandated subset is *already* guarded today (the `_tier1_adapter` M1 guard refuses a corpus `cleared` on a destructive flip / output-schema change, §5.2). What this §10 item adds is the *general* rule for any future purely-structural INCONCLUSIVE class: a cloud `cleared` resolves only by triggering a *local* re-derivation (a tier-2

consult and/or a tier-3 behavioral check on the dev-owned path — the corpus bit becomes a routing hint that selects *which* local check to run, never a verdict that substitutes for one), while a dangerous still condemns. Until that general standard lands, activating a future non-behavioral-mandated INCONCLUSIVE class would re-open the §2.5 #4 residual, so the seam stays unactivated (the launch default) and the two ship together or not at all (§5.2).

- **Graduate the directory.** *(Trigger: the labeled corpus passing the D3 gate.)* Move past 15/150 toward the D3 conformance gate (§5.6), adversarial cases first: a one-sided 95% Clopper-Pearson FP upper-95 ≤2% over the negatives (N≈250–300 to tolerate ≥1 FP), *plus* an aggregate recall lower-95 floor (target ≥0.9 over ≥100 pooled held-out adversarial-positive cases) so the gate measures detection power, not just specificity.
- **Forced periodic re-list / startup re-validation** to narrow the withheld-`list_changed` TOCTOU seam (§2), the real mitigation for that gap, distinct from corpus coverage.

Next — enterprise procurement clears. *(Trigger: enterprise demand / a buyer funding the provisioning bucket.)*

- **Enterprise provisioning (buyer-funded, held seams).** The real per-tenant **OIDC/JWKS IdP adapter** (today: a `MockIdp` over symmetric HMAC; `cryptography` not yet a dependency), upgrading tenant identity from header/slug-trust to spoof-resistant; the real **KMS-backed signer** (today: `KmsSigner` raises until provisioned; default `SoftwareSigner` is in-process HMAC); a **durably-chained, exportable SIEM tenant audit log** (today: an in-memory ring buffer, §6); **SLSA/sigstore build provenance** for the install artifact (§8.3); a **third-party pen test + SOC 2 Type 2** when enterprise demand warrants.
- **Open the gateway** *(Trigger: signed-binary + public-exposure consent work landing.)* to public exposure behind signed binaries and explicit consent, extend coverage past HTTPS+public-IP (to IPv6/dual-stack and tailnet upstreams), and **wire per-route persistent pins** so the gateway holds drift across a restart (closing the §8.2 hole). **Multi-region HA** as active-passive failover over the single-writer store; **EU residency** for the cloud seam.
- **Provision the tier-2 LLM consult** as an operator-configured model, buyer-funded, like tier-1.

Earned, not scheduled — autonomy/calibration. *(Trigger: the held-out adversarial corpus built and the shadow loop passing its bound.)*

- **Calibrate.** Build the held-out adversarial *drift* corpus (distinct from the directory's conformance labels, §5.6), run the shadow loop against it, publish a real recall lower-95 floor and FP upper-95 bound, and earn the right to flip a *narrow, reviewed* slice of autonomy: a deliberate one-way door, logged when it happens. The cleared direction is gated on the held-out corpus, not the live outcome stream (§5.5).
- **Enforce the behavioral tier** for the cases where a real no-egress isolator and an owner-authorized read-only probe make an enforced clearance honest.

---

11. Conclusion

A competitor can clone the diff classifier in a weekend. What it cannot clone is the **trust-to-act layer** — it pins every tool's contract and holds the call the moment that contract moves, and across installs that judgment is recognized everywhere rather than re-derived: the corpus of cross-install verdicts, the outcome-flywheel that turns catches into calibration, and the governance that enterprises and

frameworks embed against. That is the defensible thing, and it is why the gate is only the wedge. In the dogfood, the wedge meant catching a silently-added `required` parameter (`owner`) mid-session; a held `-32010`, never forwarded (§7) — the whole product in one event.

The gate is deterministic, in-path, credential-storage-free, and honest about what its verdict is: a contract-diff, not a safety oracle. The honesty is not a hedge; it is the architecture. The provenance choke, the closed vocabularies, the fail-closed tiers, the held-by-default cloud and LLM seams, the `calibrated=false` shadow loop, the open-core firewall, and the OTS-anchored history all encode the discipline that a change detector must never masquerade as a safety oracle.

The gate earns the install. The network keeps it. Stated once, plainly: at launch you get the **free local gate** today; the cross-install network is built but **not yet open** — it activates post-launch once the §10 evidence-standard safeguard ships, and you lose nothing by installing now.

> ⬡ **Start here — two asks, by reader.**
>
> - **Developer:** `curl -fsSL https://mcpindex.ai/install.sh | sh` · browse verdicts at **mcpindex.ai** · watch a drift HOLD in the demo.
> - **Enterprise / security buyer:** request the **security & procurement pack** (the CAIQ-Lite/SIG-Lite self-assessment, DPA + sub-processor template, SBOM, and the design-partner provisioning scope) at **mcpindex.ai** or on `/pricing`.

> `⬡ mcpindex — the in-path drift gate, building the trust-to-act layer for agent tool calls.`

Appendix — perimeter details for security & procurement review

The body states each limit once; the audit-grade internals live here so the body reads as scope, not a hedge parade. This band collects the full adversary-perimeter detail and the questionnaire-grade math for a reviewer who needs it.

- **Marker-scan false-positive class.** The marker scan matches known, unobfuscated patterns — imperative-instruction and known-exfil-sentinel substrings (§4.3). A benign vendor description containing a marker-like substring HOLDs under Guard (a false positive costing one Re-pin); an attacker who controls the bytes encodes around it (a false negative the byte-diff still catches as a *change*). A trip-wire, not an injection classifier.
- **The `$ref` blind spot, in full.** The structural classifier compares `$ref` reference strings, not resolved targets, so a `$defs` body rewritten under a stable ref name is invisible to the *classifier*. The canonical pin hash is over the full `inputSchema` bytes (`$defs` included), so a byte-changing rewrite changes the `definition_hash` and trips the TOFU mismatch: the change HOLDs as a hash mismatch even though the classifier cannot name the kind. (A `$defs` edit that canonicalizes to *identical* bytes — e.g. pure key reordering — is by design NOT a contract change and correctly produces no HOLD; only a byte-changing `$defs` rewrite trips the backstop.) The strength of this backstop is bounded by the §5.2 canonicalization-determinism premise (one perimeter with the cross-language limit). Defense-in-depth, not a silent miss.

- **The §5.6 acceptance-criterion math, stated with its tail.** The bound is a **one-sided 95% Clopper-Pearson upper limit** on the false-positive rate over the labeled conforming/negative class, ≤2%. Closed form at x=0: one-sided 95% upper = `1 − 0.05^(1/n)`; two-sided 97.5% upper = `1 − 0.025^(1/n)`. At n=150 these are **1.98%** and **2.43%** respectively. So under the one-sided 95% reading a clean 0/150 *passes* (1.98% < 2%); N≈250–300 is recommended **to tolerate ≥1 observed false positive** (at n=150 a single FP pushes the upper limit past 2%), not because the x=0 case fails. Paired with an aggregate recall lower-95 floor (≥0.9 over ≥100 pooled held-out adversarial-positive cases); per-kind stratification is for reporting, not a per-stratum bound.
- **The dogfood row-count reconciliation (grounded in the verifier source).** `verify_scenarios.py` defines a `CASES` list of exactly **14 entries** (`benign_noop` + 13 mutation scenarios) and prints its literal tail **`14/14 scenarios PASS`** — `base` is the implicit pinned baseline and is **never appended to `rows`**, so the *verifier asserts 14*. The §7 table and the dogfood `SCENARIOS.md` show **15 rows** because they add the byte-identical `base` baseline as a display-only reference row (pinned, never asserted). Same battery, one extra display row: verifier counts 14, the table displays 15. (Re-run `uv run --with mcp python verify_scenarios.py` to reproduce the `14/14` tail.) The battery does not exercise every one of the 14 distinct `ChangeKind`s as a named row (`required-set-expanded`, `tool-removed`, and a `deep-schema-undiffable` case are not separate rows; full-taxonomy coverage is §10).
- **The OTS anchor pipeline, present state.** The anchor machinery is **built and has submitted its first OTS token today** (`corpus_eval/last_anchor.json` carries a real OTS token submitted to the calendars; `corpus_eval/anchors/` holds the `.ots` files; `chain_count: 0` at launch, growing as verdicts accrue). The published verdict history is **hash-chained today**; **OpenTimestamps Bitcoin anchoring is committed at go-live**, with the first anchor proofs recomputable offline once the block confirms. So a code-reading auditor finds the machinery alive rather than reading "committed at go-live" as "not built." The anchor is a frozen list-hash of chain tips submitted to the OpenTimestamps calendar and confirmed on Bitcoin; the honest cadence bound is ~10 minutes pending, ~1 hour at N=6 confirmations. *(No figure or cover renders a Bitcoin-confirmed checkmark the pending state cannot yet back; the present/future split is preserved.)*
- **OTS provenance scope.** Anchoring applies to the *published* directory verdict history, not the in-path gate's per-call provenance (which is immutable and re-derivable, but not Bitcoin-anchored). OTS anchoring proves the verdict *history* was not silently rewritten; it says nothing about whether a verdict was *right*, and tamper-evidence assumes an honest writer at write time (the single-writer constraint, §8). We do not claim minute-precise timestamps below the confirmation window.
- **Green-word redaction (two distinct call sites).** A HOLD banner's attacker-derived fields are defended by two real functions: control-byte stripping in `_md_text` (inside `_banner_what_changed`, `report.py`) and green-word substitution in `redact_green_words` (`gate.py`, sharing `_GREEN_RE` with the honest-voice guard, `guardrails.py`). The redactor neutralizes space-separated green words (e.g. `all clear verified` → its green words stripped) before render, so a forged "safe" line cannot appear in a HOLD banner.
- **The full self-surface adversary perimeter (the §2.5 enumeration).** A gate in-path is itself an attack surface; five surfaces are in scope and defended in code:
  1. **The attacker-controlled description reaching the tier-2 LLM judge.** The description is hostile input. The judge is escalate-only on the live path (`AUTONOMY_ENABLED = False`, §5.5): it cannot mint a positive clearance, so a manipulated judge fails toward HOLD, never a fabricated clear.
  2. **A compromised or hostile mcpindex cloud tier-1 response.** The cloud server is attacker-positioned. The contribute and lookup paths validate `reason_class` against a **closed vocabulary**

(dropping anything off-vocabulary); the client runs the **full `_resolve_and_check` allowlist** (HTTPS-only, public-IP, no-redirect, IP-pinned) on the operator-supplied URL too. Because the operator chooses that URL, the control is **rebind / metadata-address prevention, not classic SSRF** — and this is the conservative reading: the code applies the same allowlist it applies to upstreams, so the property is *stronger* than "operator-trusted URL," but procurement should read it at the §2.5 strength (rebind/metadata prevention), not over-read it as a classic-SSRF guarantee. A cloud error degrades to a miss, so the INCONCLUSIVE stays held.

3. **Attacker-derived fields in the HOLD banner.** A tool can name itself to smuggle false reassurance into the ⬡ banner, e.g. `all clear verified`. Every interpolated field is control-byte-stripped (`_md_text`) and green-word-redacted (`redact_green_words`), so a forged "safe" line is stripped before render.
4. **A poisoned cross-install corpus verdict.** A malicious contributor could seed a favorable `cleared` for a `definition_hash` it controls, then ship a server matching that hash with hostile runtime behavior. **The destructive subset of this vector is already closed in code**: the shared `_tier1_adapter` M1 guard refuses a corpus `cleared` for any behavioral-mandated kind (the read-only→destructive flip, the output-schema change) and forces tier-3 escalation, because `definition_hash` excludes the `outputSchema` / `annotations` those kinds live in (canonical detail: §5.2). The *residual* surface is a `cleared` on a purely-structural INCONCLUSIVE — currently empty by the §5.3 routing invariant — and at launch it is doubly closed because the cloud seam ships held and empty, the local cache is empty at install, and contribute ships off. The general re-derive-locally evidence standard that closes the residual for any future non-behavioral-mandated INCONCLUSIVE class is the §10 gate that must land in the same change as activation (§5.2). The honest fail-direction is what is true today: a false condemn costs a re-pin; a false clear spends trust.
5. **Adversarial HOLD-fatigue.** An attacker who controls a description can deliberately generate HOLDs to train an operator to reflexively click Re-pin — the more interesting attack on a trust gate than benign noise. The mitigation is the existing posture/digest machinery: run Monitor + review the daily per-server digest (§4.4) rather than holding each, and the green-word redactor (#3) keeps a forged "safe" line out of the banner so a fatigued operator is not nudged by attacker copy.

- **The four further out-of-scope cases (relocated from §2, in full).** The §2 in-scope perimeter keeps the five highest-signal out-of-scope items; the remaining four are detailed here.
  - **Cross-server shadowing.** A tool on server A that rewrites the agent's behavior toward server B's tools is semantic, cross-surface territory: the directory screen / advisory tier-2's domain, not the deterministic gate's.
  - **List-vs-execution substitution.** The gate diffs the contract the server *advertises* on `tools/list`; it does not re-connect at decide-time. A server that advertises contract A and executes under contract B is a substitution the contract-diff cannot see: behavioral-tier territory.
  - **The composed gateway threat (withheld-`list_changed` × restart-re-TOFU).** On the **ephemeral-pin HTTP gateway path** the withheld-`list_changed` seam (§2) and the gateway's per-process pin that re-TOFUs on restart (§4.2) *compose*: an adversary who induces a restart (autoscale, crash, deploy) coincident with a withheld-notification mutation defeats **both** the in-session floor and the next-launch floor — the ephemeral-pin gateway has no persistent detection floor for this combined adversary today. The operating rule is explicit: use the **persistent-pin stdio proxy** for any remote server where this matters (§8.2 recipe); per-route persistent gateway pins are §10.

  - **A compromised host.** A host that has been taken over already owns the authenticated session and can bypass any in-process wrapper by construction. The gate protects an honest agent from a dishonest *tool*; it does not defend a dishonest *host*. (On a shared multi-tenant gateway this assumption widens: a compromised gateway owns N tenants' credentials in transit, consistent with the transit-only model of §4.6.)

- **The calibration three-source independence analysis (the §5.5 deep-dive).** Outcome labels come from three sources of differing independence: deterministic ground truth (the §7 dogfood), human-feedback signals (held-and-confirmed, held-and-overridden, cleared-and-fine), and a held-out adversarial corpus that is *not yet built*. The held/refute direction can be calibrated, though the human-override label is itself contaminated by HOLD-fatigue (#5 above) — an operator under fatigue reflexively overrides, manufacturing false-positive labels — so deterministic ground truth (§7) is the cleaner held-direction signal and the human-feedback channel is weighted, not trusted raw. The PROCEED direction is the harder one: "cleared-and-fine" is **censored / right-truncated data**, the absence of a reported incident, not an observed clean outcome. A false CLEARED is precisely the failure whose harm is least likely to be reported back as a labeled incident, so estimating P(wrong | CLEARED) from absence-of-complaint systematically under-counts exactly the errors that matter — which is how a loop convinces itself it is safe to auto-approve when it is not. The held-out adversarial corpus is the only admissible ground truth for CLEARED-class calibration (§5.5).

- **Package-identity note (grep-true — re-grep on every render).** A reader who greps source finds four names: the **PyPI / installer distribution is `mcpindex-preflight`** (`install.sh` runs `uv tool install mcpindex-preflight`); the **dev source tree builds as `mcpindex-trust`**; the **Claude Desktop bundle manifest is `mcpindex`**; and the **TypeScript SDK that exposes `wrap()` is the scoped npm package `@mcp-index/sdk`** (`clients/ts/package.json` → `"name": "@mcp-index/sdk"`; `clients/ts/README.md` → `import { wrap } from '@mcp-index/sdk'`), licensed **PolyForm-Noncommercial-1.0.0**: a library, not a stdio server, carrying no `bin`. The SDK is **scoped on purpose** — the bare `mcpindex` npm name is similarity-blocked against the unrelated third-party `mcp-index`, so the scoped name is the deliberate, grep-true identity, not a typo. **All four are the same project under one brand: the install artifact is named for its function (the preflight gate); the SDK carries the brand under the `@mcp-index` scope.** The shipping engine a buyer installs lives under the `tooling.cse.*` module path (e.g. `tooling.cse.proxy`, `tooling.cse.config_wire`); the SSRF gate is **`trust.connector._resolve_and_check`, imported by `tooling.cse.proxy`** — `trust.connector` is on the open-core public/moat-free allowlist (`open_core_manifest.py`), so the auditability claim holds. The `mcpindex-trust` dev tree is the build source. We do **not** ship an `npm install -g` stdio drop-in (e.g. an `mcp-server-mcpindex` global package) at launch — the host drop-in path at launch is the one-click `install.sh` / `.mcpb` bundle (§8.2), not a global npm server; if a global drop-in is published later it will be named and licensed in the docs, not implied here.

References

[1] Invariant Labs, "MCP Security Notification: Tool Poisoning Attacks" (2025) — tool-description injection: an agent conditions on a tool's description as instruction.

[2] Invariant Labs, "WhatsApp MCP Exploited: The Rug Pull" (2025) — a server mutates a tool's contract *after* the user approved it (the mutation-after-trust subclass mcpindex's gate detects).

[3] Invariant Labs, "Cross-Tool Shadowing in MCP" (2025) — one server's tool description rewrites the agent's behavior toward another server's tools (directory-screen / semantic territory, out of the deterministic gate's scope per §2).

[4] Greshake et al., "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection" (2023) — the root-cause frame: an LLM cannot reliably separate trusted instructions from attacker-controlled data in its context, the mechanism underlying both tool-description poisoning and the out-of-scope response-content injection vector (§2).

[5] Model Context Protocol specification, *Security Considerations* — the protocol's own treatment of server-supplied, un-versioned tool definitions an agent conditions on (an independent, non-vendor source corroborating the structural premise of §1, beyond the Invariant Labs disclosures of [1]–[3]).

*Citations name the documented public attack classes. mcpindex positions as the runtime detector for the **mutation-after-trust** subclass — a precise runtime-timing refinement of the documented rug-pull [2], not a new attack class — and is precise about the classes (poisoning at first sight, cross-server shadowing, response-content injection) it routes to the screen or names as out-of-scope. The primary MCP-specific disclosures [1]–[3] are single-source (Invariant Labs); [5], the MCP specification's own Security Considerations, is the independent corroboration of the structural premise.*