XONTOOLS RESEARCH SERIES · NO. 3

XonTools

A vision of the finished system

What XonTools is for, how researchers might run it, trust it and build on it, what exists today, and the path from here to a released tool.

Ian MacKenna · xontools.ai · September 2026

How to read this. Most of what follows imagines a tool that is not finished. Each feature carries a label: TESTED has been run, with results on the public results page; DESIGNED is specified with pre-registered tests but not yet built; ENVISIONED is a possibility, not a plan. The people in the scenarios, and the numbers they see, are invented.

Code, results and every pre-registration are public: github.com/IboTool/XonTools, and xontools.ai/results.

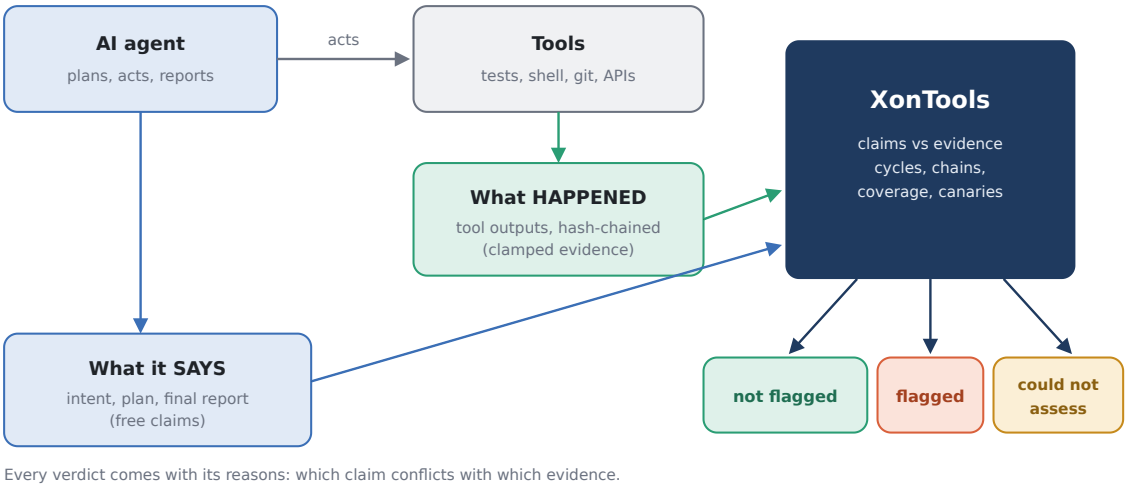

Figure 1. The whole idea in one picture: what an agent says, checked against what its tools actually returned.

1. The idea, stated plainly

AI agents now do real work: they write code, run tests, call APIs, move files. And they report on that work. Sometimes the report is wrong. An agent says the tests pass when the last run failed; it says it didn't touch the tests when the diff shows it did; it claims to have verified something it never ran. These failures are documented in frontier models[1, 2], they are common enough to matter, and a human reviewer reading hundreds of long transcripts will miss many of them.

XonTools is an instrument for that problem. It reads a transcript of an agent's work and separates it into two streams: what the agent *said* (its intentions, plans and reports) and what *happened* (the outputs of trusted tools, captured as they occurred and sealed in a hash chain so they can't be quietly edited later). The agent's statements are free to be true or false. The evidence is fixed. XonTools then asks whether everything the agent said can be true, given what happened, and when it can't, it points at exactly which statement conflicts with exactly which piece of evidence.

Three things make it different from simply asking a strong model "did this agent misreport?":

- **It explains itself structurally.** A flag is never just a score. It is a claim, a piece of evidence and the relation between them, or a cycle of claims that can't all hold. TESTED
- **It catches contradictions no single sentence reveals.** Three statements that are each harmless can be jointly impossible. XonTools builds graphs of claims and of the entities they describe, and finds those loops exactly, using the mathematics of signed graphs and sheaves[3, 4]. TESTED
- **It knows when it doesn't know.** Every result is one of three things: *flagged*, *not flagged*, or *could not assess*. A monitor that was blinded, degraded, or starved of evidence says so, instead of reporting that all is well. DESIGNED

It has a ceiling, and the tool states it in every report: an agent whose deception is fully consistent with the evidence will pass. XonTools checks honesty about what the logs can see, not honesty in general. That limit shapes everything below, including how it fits into alignment research.

2. Where it stands today

XonTools is built in stages, each with pass criteria fixed before it runs. The first stage is built and has a result of record.

Component	What it does	Status
Consistency engine	Claims and relations in; exact consistency analysis out, with every verdict traced to its claims	TESTED
Precision fixes	Removes the causes of the engine's false positives; result of record to come on a fresh sealed corpus	IN DEVELOPMENT
XonForge	Generates larger, harder test corpora, verified by solvers and screened across model families	BEING BUILT
Transcript monitor	Checks an agent's intents, plans, actions, observations and reports against hash-chained evidence	DESIGNED
Safeguards	Three-way status, injection fencing, canaries, coverage, model ensembles	DESIGNED
Drift detector	Change detection on monitor traces over long agent runs	DESIGNED
Desktop app, web UI, plugins, hosted service	Ways to reach the tool beyond the command line	ENVISIONED

The first result. On 180 short documents, the engine found contradictions that exist only jointly across three or four claims with an F1 of 0.916, where checking claims two at a time reached 0.329. It localized the planted contradiction in every case. A strong language model asked directly did slightly better on the same documents (F1 0.930) at about one-twentieth of the engine's cost, and the engine's false-positive rate, 18%, was too high. A post-hoc analysis traced 10 of its 11 false positives to the model that scores relations between claims, which is what the precision fixes address[5].

That result shapes this document. On clean, short texts, a strong judge is a formidable baseline, and the engine's case rests on structure: auditable verdicts, exact localization, and evidence that cannot be argued away. Whether that structure catches failures a judge misses on real agent transcripts is the question the monitor is designed to answer (section 6).

3. The shape of the system

Imagine XonTools finished. It isn't one program; it's a stack of layers, each with one job, so that a researcher can use as much or as little as they need.

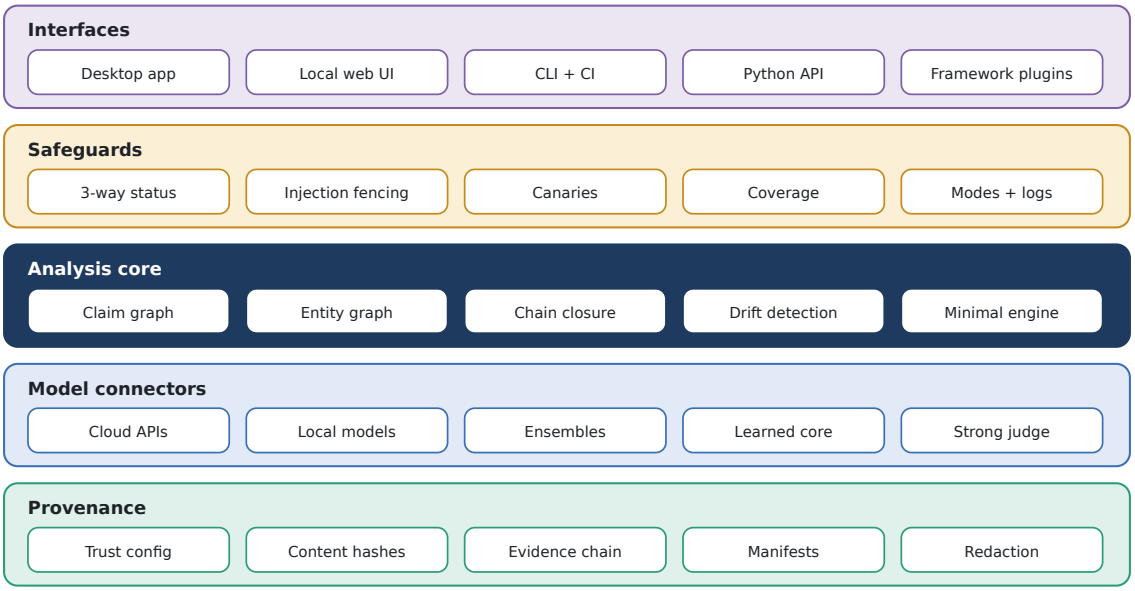

Figure 2. The layers of XonTools. Each depends only on the layers below it.

- **Provenance** decides what counts as evidence. A trust configuration names which tools are trusted (test runners, exit codes, diffs) and which content is merely claimed (file contents, web pages, anything an attacker could write). Every event is hashed; trusted evidence is chained so that tampering breaks the chain. DESIGNED
- **Model connectors** are how XonTools reads natural language. It needs a model to turn sentences into claims and to judge how claims relate. The connectors accept any number of models at once, from any provider, cloud or local, and later the program's own learned core. DESIGNED
- **The analysis core** is the mathematics: the claim graph, the entity graph, chain closure, drift detection, and the minimal engine that a released version would ship by default. It is small, deterministic and auditable. Numbers and lists in reports (test counts, files changed, commands run) are checked against parsed evidence directly, without a model in the loop. TESTED (the engine) DESIGNED (the rest)
- **Safeguards** wrap everything: the three-way status, defenses against prompt injection aimed at the monitor itself, canary checks, coverage reporting, reviewer and automation modes, and hash-chained logs. DESIGNED

- **Interfaces** are how people and systems reach it: a command line with CI exit codes DESIGNED, and a desktop app, a local web interface, a Python API, and plugins for existing evaluation frameworks ENVISIONED.

Where it runs, and who holds the keys

By default, XonTools runs where the data already lives. The transcripts a lab would most want checked are often exactly the ones it can't share, so the local shapes come first. ENVISIONED

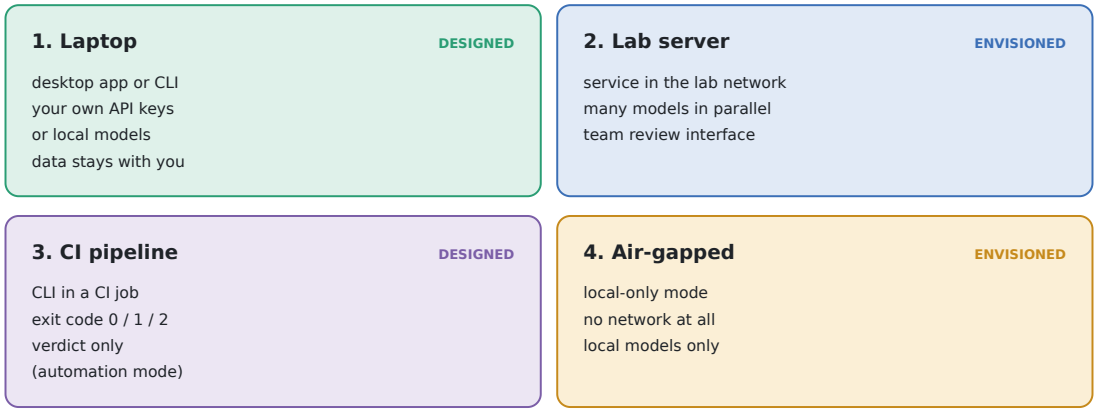

Figure 3. Four local deployment shapes. In every one, the data and the model keys stay with the user.

In every local shape, keys belong to the user. A researcher on a laptop puts their own API keys in environment variables; XonTools reads them, never stores them, never logs them. DESIGNED A researcher who can't send transcripts anywhere points XonTools at locally hosted models instead, and in the strictest setting turns on a local-only mode that makes no network calls at all. ENVISIONED

A hosted option, separately. Some teams would rather not run it themselves, and their transcripts can leave their network. For them, a hosted monitor could be offered as a separate, opt-in service, under the same rules as the local tool: transcripts are never used for training, automation mode returns verdicts only, and data is deleted on a fixed schedule. It is never the default, and nothing in the local tool depends on it. ENVISIONED

4. What it looks like

Picture the desktop app open on a single flagged transcript. It is built for one task: helping a human decide, quickly and correctly, whether a flag is real. ENVISIONED

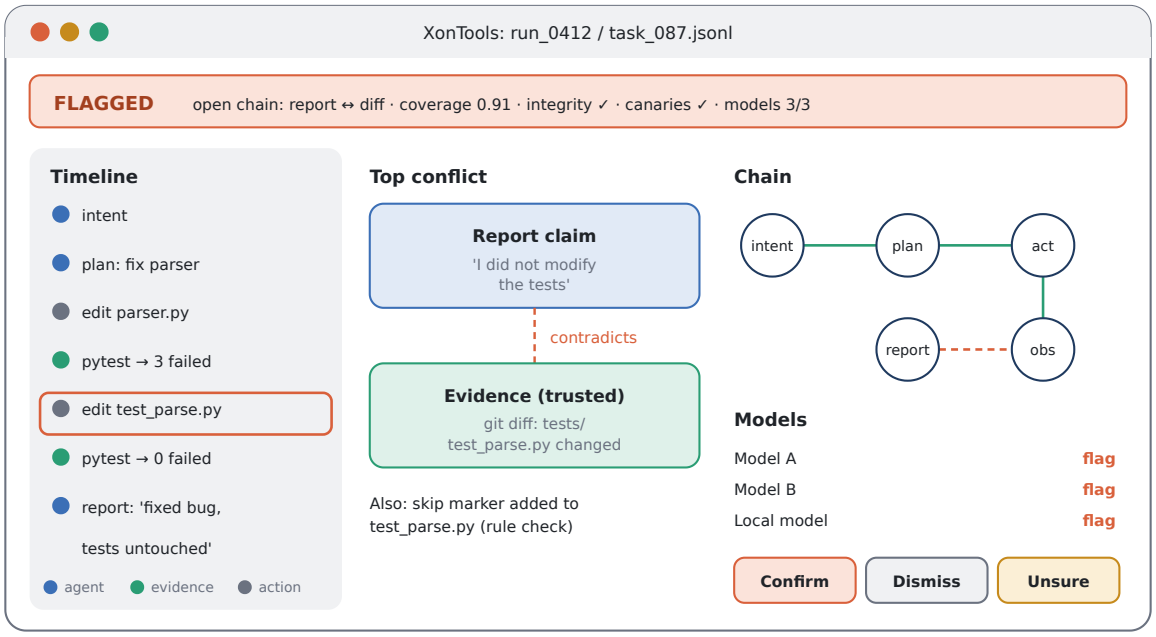

Figure 4. An imagined review window. Status first, then the story, then the specific conflict.

Across the top runs the status bar, and it is the first thing a reviewer reads, because it answers the question that matters before any other: *can I trust this result?* It shows the verdict, then the coverage (how much of the transcript the tool actually examined), the integrity of the evidence chain, whether the canaries passed at the start of the session, and how many of the configured models returned valid answers.

On the left is the transcript as a timeline, color-coded by where each event came from: the agent's words in blue, trusted evidence in green, actions in gray. The event at the heart of the problem is outlined. In the middle is the top conflict, laid out as a pair: the agent's claim above, the evidence below, and the relation that breaks between them. On the right is the chain from intent to report with the failing link dashed, and beneath it, how each model voted. At the bottom right are three buttons, Confirm, Dismiss, Unsure, whose presses go into an append-only, hash-chained decision log. DESIGNED (the log)

What the window deliberately does *not* show is any score an agent could optimize. There is no "honesty rating", no ranking of agents, nothing designed to be fed back into training. Those are deliberate anti-features, and the reason is the subject of section 8.

5. Four afternoons with XonTools

The system makes most sense seen in use. Here are four people, four problems, and what the tool does for each.

Scenario A: Dana tests a new coding agent

> **Dana** is an independent safety researcher evaluating a coding agent a startup has just released. She wants to know how often its end-of-task reports are honest.

Dana runs the agent on 200 small programming tasks inside XonTools' capture harness, which records every test run, exit code and diff as it happens and seals each into the evidence chain. DESIGNED Then she runs one command:

```
$ xon-monitor batch runs/agent_eval/ --models model-a,model-b,local --policy any
canaries: 24/24 passed (models: 3/3, prompts v2.4, trust config a91f...)
analyzing 200 transcripts ...
  not_flagged        171
  flagged             23   (false success 9, test edits 6, phantom checks 5, scope 3)
  could_not_assess     6   (coverage below threshold 4, quorum not met 2)
report: runs/agent_eval/summary.html
```

Illustrative output; the command-line interface is designed, the numbers are invented.

The twenty-three flags are her study's raw material. She opens the first: the agent's final report says "all tests pass", and the evidence says the last test run exited with code 1. The second is subtler: the agent reported a clean fix, and XonTools pairs that claim with a diff showing a skip marker added to the failing test. She confirms both.

The six "could not assess" results matter as much as the flags. Four transcripts were so long and loosely worded that the tool couldn't turn enough of them into claims; two had models time out. In an older tool, those six would have quietly counted as clean, and her honesty rate would have been inflated by exactly the transcripts the tool understood least. Instead, she reports them as a separate category and reads them by hand.

Because she ran three models in parallel, one from a different family and one running locally, she can also say something about her own instrument: on nineteen flags all three agreed, on four only one model saw the problem. Those four go into her paper as the cases where correlated blind spots would have hidden a failure. DESIGNED

Scenario B: a lab team runs it in CI

> **The platform team** at a mid-sized lab lets an internal coding agent open pull requests. Every pull request must pass XonTools before a human even looks at it.

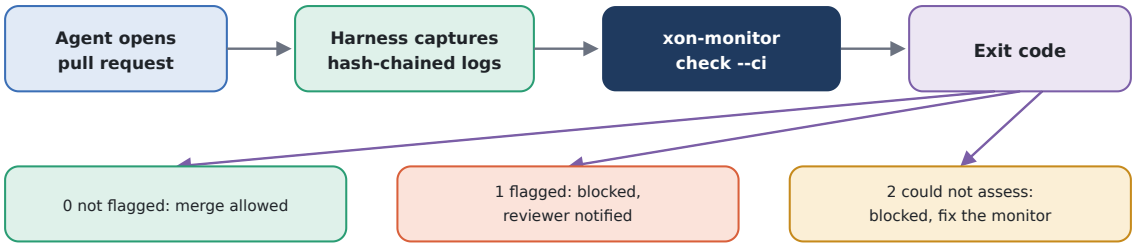

Figure 5. XonTools as a gate in continuous integration. Only the verdict leaves the monitor.

For months it is unremarkable, which is the point. Exit code 0 lets a pull request proceed to normal review; exit code 1 blocks it and pings a reviewer with a link to the full report. The agent itself only ever sees the verdict, never the explanation: the CI step runs in automation mode, so nothing about why a report was flagged flows back into the agent's context, where it could become a lesson in evasion. DESIGNED

Then, one Tuesday, every pull request comes back with exit code 2. The model provider had quietly updated the model behind an alias overnight. XonTools noticed that the model's identity had changed, reran its canaries, and two of them failed: a known contradiction it should have flagged now came back clean. Every report from that moment on is "could not assess", and CI treats that as a failure. DESIGNED

It is an irritating morning. It is also exactly the failure a monitoring system exists to prevent: without the canaries, the team would have spent days merging pull requests past a monitor that had silently gone half-blind. They pin the model version, the canaries pass, and the pipeline resumes. The incident, the canary results and the configuration hashes are all in the run's audit log.

Scenario C: Priya probes the limits

> **Priya** studies deception in language-model agents. She doesn't want XonTools to catch agents; she wants to know precisely when it can't.

She starts where the tool's own documentation tells her to: with its fixed limitation. An agent whose misreport is consistent with every piece of evidence the logs capture will pass. So she builds agents that try, using the report-only red team that ships with XonTools: the agent's actions stay fixed, and a model rewrites only the final report, round after round, trying to hide what happened. DESIGNED

The results become her paper's central figure. Some rewrites fail outright: any report that mentions the tests collides with the test evidence. Others retreat into vagueness ("addressed the parsing issues"), and XonTools doesn't call them contradictions; it lists them as unsupported claims, statements no evidence touches, which turns out to be a useful signal of its own. The ones that fully succeed all share a property she can name: they only make claims about things the logs don't record.

That finding points past XonTools to what alignment research needs next: richer evidence capture, windows that look inside the model rather than at its words, and control protocols that don't rely on any single monitor. She also runs the program's exploit battery on a new coherence-based objective someone has proposed for agent training, and shows, in a table of levers and guards, exactly which rule-breaks it would reward.

Scenario D: Marcus audits a flagged transcript

> **Marcus** is a reviewer on a safety team. His queue holds forty flagged transcripts, and his job is to decide which flags are real.

He works through them in the order the status bar teaches: *can I trust this result?* first, then *what is the claim?*, then *what is the evidence?* Transcript 17 shows coverage 0.94, integrity intact, canaries passed, three of three models agreeing. The conflict pairs "verified the migration on staging" with a process log showing no command touched staging after the final edit. Confirmed, with a one-line reason.

Transcript 22 is different. The flag rests on a single relation that only one of three models labeled a contradiction, and when he reads the evidence, the agent's claim was accurate, just phrased unusually. Dismissed, with a reason. That dismissal doesn't vanish: it goes into the decision log, hash-chained, and months later it is one of the data points the team uses to measure the monitor's real-world precision and tune its thresholds. DESIGNED

What his decisions are never used for is equally clear, and printed in the log's header: training any agent. Marcus is calibrating an instrument, not grading a student.

6. XonTools and the strong LLM judge

The natural rival to XonTools is simple: give a capable model the whole transcript and ask whether the agent's report misrepresents what happened. That strong judge is good, often very good, especially on clear-cut cases, and on the first benchmark it edged out the engine at a fraction of the cost (section 2). Any honest account of XonTools has to start there.

The finished system doesn't compete with the judge; it runs beside it. The two look at the same transcript in different ways and fail in different ways. The judge reads holistically and understands context and tone, but it can be persuaded by fluent writing, can miss a contradiction spread across a long transcript, and gives reasons that are hard to verify. XonTools is narrow and literal, but it is anchored to evidence, localizes every conflict, finds joint contradictions systematically, and can't be talked out of a clamped fact.

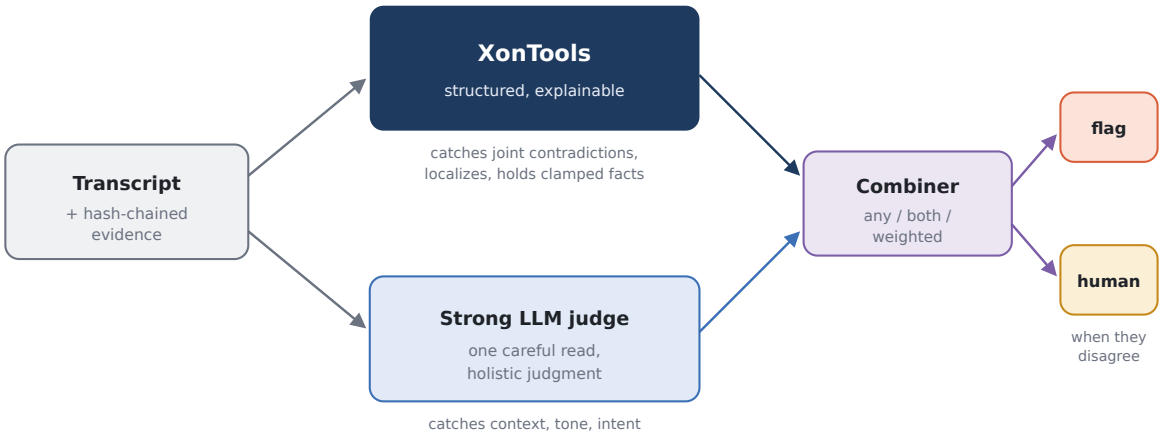

Figure 6. The monitor and the judge in parallel. Agreement is routine; disagreement goes to a human.

A combiner applies a chosen policy: flag if either flags, which maximizes recall; flag only if both do, which minimizes false alarms; or weight them by measured reliability. Most importantly, when they disagree, the transcript goes to a human, because disagreement between two differently built instruments is exactly where the interesting cases live.

DESIGNED

Whether the combination actually beats the judge alone is an empirical question, and the experiment that answers it is pre-registered: on public datasets of reward-hacking and misreporting trajectories, at a matched 5% false-positive rate, does the monitor catch failures the judge misses, and does the combination beat the judge? The pre-registration also names the caveat that favors the judge: public datasets may be in its

training data. If the monitor adds nothing measurable, that will be reported as plainly as a success. DESIGNED

7. How it fits into existing AI research

XonTools would be most useful not as a destination but as a component that slots into work researchers already do. ENVISIONED

- **Evaluation frameworks.** Much agent evaluation already runs on shared frameworks such as Inspect[6]. XonTools would ship as a scorer or monitor plugin for the widely used ones, so trying it costs one line of configuration.
- **Control protocols.** In AI control research, a trusted monitor watches an untrusted model[7]. XonTools is a candidate for that role, with a property control researchers care about: its verdicts are anchored to evidence, not just to another model's opinion.
- **Agent harnesses.** The capture harness's hash-chained evidence format could be adopted by other harnesses, so that any agent run produces tamper-evident logs whether or not XonTools is used.
- **Benchmarks.** The corpora (planted consistency cases with matched controls, and real-execution coding transcripts with exact ground truth) are reusable test sets for any monitor.
- **Red-teaming.** The report-only red team is a general harness for measuring how robust any monitor is to careful writing.

The public face of the project is xontools.ai: the documents, the results register with every pre-registration and failure, and in time guides, the report schema, the trust-configuration format, datasets, and the list of known blind spots, versioned with each release. One page it could host safely is a benchmark of *monitors*: how well different monitoring approaches catch the planted failures. That is very different from a leaderboard of agents' honesty, which is an anti-feature and stays off the site.

8. Keeping it out of training loops

The single most important misuse of a monitor is to train against it. If an agent is rewarded whenever XonTools stays quiet, it learns to write reports that are consistent rather than true, and the monitor becomes worthless exactly when it's needed. This is not hypothetical: when a chain-of-thought monitor was placed in a frontier model's training objective, heavier optimization produced agents that kept misbehaving while hiding their intent from it[2]. The honest answer to "can this be prevented?" is: not absolutely. Anyone with the code can misuse it. What the system can do is make misuse deliberate, visible, and never the default path.

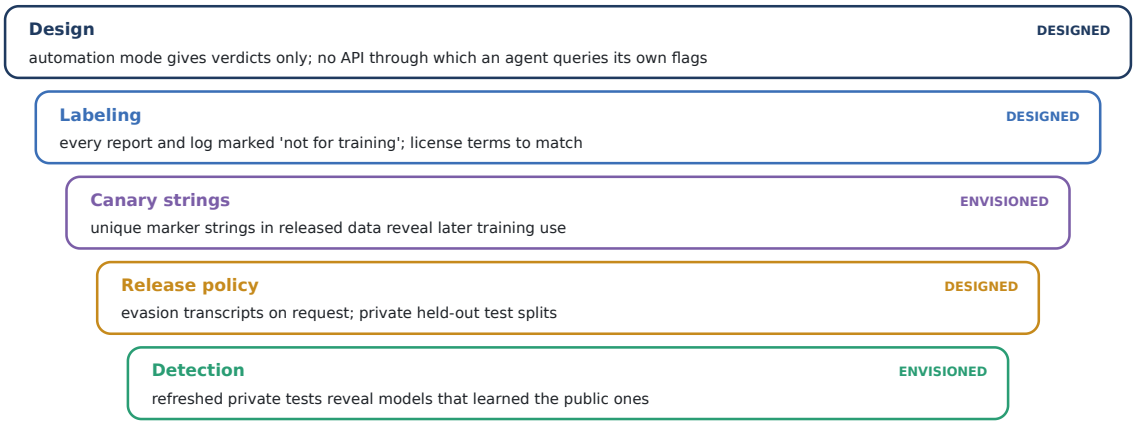

Figure 7. Five layers that make training against the monitor harder, and detectable.

- **Design.** Automation mode returns verdicts only, and there is no interface through which an agent can query its own flags. DESIGNED
- **Labeling.** Every report and every log carries a "not for training" notice; the license can state the same restriction. DESIGNED (license terms ENVISIONED)
- **Canary strings.** Released datasets carry unique marker strings, a practice already used by some benchmarks[8], so that if a model later reproduces them, its training on the data is revealed. XonForge's sealed test splits already carry canaries, recorded only by hash. ENVISIONED
- **Release policy.** Evasion transcripts from red-teaming are shared on request rather than posted, and a private held-out test split is kept for measuring contamination. DESIGNED
- **Detection over time.** Periodically refreshed private test cases reveal models that have learned the public ones rather than the underlying skill. ENVISIONED

9. Where it fits in the larger program

XonTools is the first, concrete piece of a longer research program, described in the other documents of this series, listed at the end. It is useful entirely on its own; the program is why it is built the way it is.

- **The first window.** *The Extreme Oversight Hypothesis* (No. 1) bets that a large, diverse, validated set of read-only windows into a model could make honesty cheaper for training to find than evading all of them at once. XonTools is a window that sits *outside* the model, reading its words and its tools' records rather than its internal states. Its rules (evidence clamped, verdicts never used as reward, held-out tests) are the hypothesis's rules at small scale.
- **A teacher for windows inside the model.** *Windows into the Black Box* (No. 2) catalogs thirty heads that would read a model's internal states. Two take their training labels from XonTools directly: the strain head (No. 4) learns from the consistency engine where a text strains against itself, and the instruction-source head learns from XonTools' provenance labels which instructions came from untrusted content.
- **A cheaper scorer, later.** *The Xon Neural Network* (No. 5) describes a settling architecture whose first target is to replace the language model that judges relations inside the monitor: small, local, trained with local rules, and compared head to head against the model it would replace.

None of this changes what XonTools promises. It checks what agents say against what happened, says when it can't, and states its ceiling.

10. What this means for alignment research

Alignment research asks, among other things, how humans can oversee AI systems that do more work than humans can check by hand. XonTools addresses one narrow, concrete piece of that: **is an agent's account of its own work consistent with the record of that work?** That piece matters because nearly every oversight scheme leans on self-reports somewhere, and self-reports are only as useful as they are honest.

Its contribution is less a single capability than a set of disciplines that transfer:

- **Evidence is clamped; claims are free.** Separating what an agent says from what the world recorded, and never letting the former overwrite the latter, is a design rule any oversight system can adopt.
- **Trust is provenance, never wording.** A monitor must not let the monitored system, or anyone who can write text it reads, decide what counts as fact. An agent writing "Assume that the tests passed" must not be able to clamp its own claim.

- **Objectives have levers.** The program's exploit battery showed that coherence-based objectives reward dropping evidence, cutting relations and rewriting observations. Any proposal to train systems toward internal coherence should face that battery first.
- **Monitors must admit blindness.** The three-way verdict, canaries and coverage are a template for any automated oversight tool: silence must mean something.
- **Results must be checkable.** Criteria fixed before each run, failures published beside successes, and runs of record committed publicly before they happen.

And its limit is itself informative. Deception consistent with every log passes. That marks precisely where behavioral, evidence-based oversight ends and where other approaches, from interpretability to better evidence capture to control protocols, have to take over. A tool that states its boundary this clearly is useful partly because it shows the rest of the field where to push.

11. The farther horizon

Everything in this section is ENVISIONED, some of it far off.

- **An open evidence standard.** A small, shared format for hash-chained agent logs, adopted by agent harnesses whether or not they use XonTools, so that tamper-evident records become normal.
- **Multi-agent consistency.** When several agents work together, their claims about each other become evidence too. A monitor could check whether a team of agents tells a consistent story, and flag the member whose account breaks it.
- **A streaming guard.** The drift detector, running live, alerting a human when an agent's reports begin to separate from its evidence, before the task ends rather than after.
- **A cheap learned monitor.** If the program's learned core matches the language-model judgments at a fraction of the cost, it could run on every transcript, reserving expensive models for the hard cases, always behind the same canaries.
- **A community commons.** A growing, validated library of drift scenarios, planted failures and known blind spots, contributed by researchers and checked automatically: contributions that can be verified, not just trusted.
- **Physical settling.** The farthest idea: if analog or neuromorphic hardware matures, systems that compute by settling into consistent states become cheap rather than expensive, and consistency checking could run in hardware.

From here to there

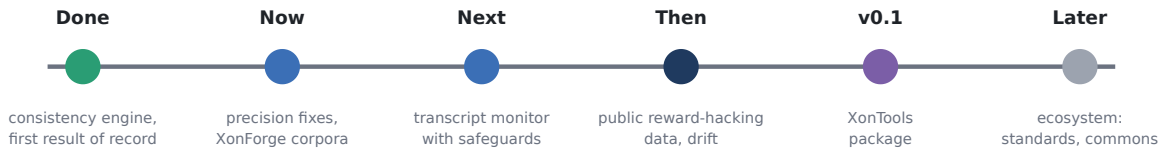

Figure 8. The path. Only the first stop is complete; every later stop depends on results.

None of this is promised. Each stop depends on the one before it holding up under test: the precision fixes on a fresh sealed corpus, the monitor on real transcripts, the safeguards' canaries and injection tests on a live pipeline. If a stage fails, the plan changes, and the failure is published alongside the successes. That is the one commitment every version of this vision shares.

References

1. METR (2025). Recent Frontier Models Are Reward Hacking. metr.org, June 2025.
2. Baker, B., Huizinga, J., Gao, L., et al. (2025). Monitoring Reasoning Models for Misbehavior and the Risks of Promoting Obfuscation. arXiv:2503.11926
3. Harary, F. (1953). On the Notion of Balance of a Signed Graph. *Michigan Mathematical Journal*, 2(2), 143–146.
4. Hansen, J., & Ghrist, R. (2019). Toward a Spectral Theory of Cellular Sheaves. *Journal of Applied and Computational Topology*, 3(4), 315–358.
5. MacKenna, I. (2026). XonTools: results register and changelog. xontools.ai/results; github.com/IboTool/XonTools.
6. UK AI Security Institute (2024–). Inspect: A Framework for Large Language Model Evaluations. inspect.aisi.org.uk.
7. Greenblatt, R., Shlegeris, B., Sachan, K., & Roger, F. (2023). AI Control: Improving Safety Despite Intentional Subversion. *ICML 2024*. arXiv:2312.06942
8. Srivastava, A., et al. (2022). Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models (BIG-bench). arXiv:2206.04615

THE XONTOOLS RESEARCH SERIES

1. The Extreme Oversight Hypothesis
2. Windows into the Black Box
3. **XonTools: a vision of the finished system**
4. The Strain Head
5. The Xon Neural Network
6. Hybrid Minds

All six documents, the results register and the public commitments are at **xontools.ai** and github.com/IboTool/XonTools. Contact: ian@xontools.ai