Enabling Verifiably-Scoped Monitoring through Large Language Models and Trusted Compute

Benjamin Penchas [1] **G Zhao** [2] **Roy Rinberg** [3]

Abstract

Frontier AI labs increasingly monitor chat interactions for safety-relevant harms and to gather information on broad usage patterns, but the same infrastructure could be repurposed for surveillance, and monitored parties have no technical means to verify that a stated monitoring scope matches the actual one. We propose a two-party protocol in which a monitor and a monitored party jointly sign a Plan that is executed by an open-source LLM classifier inside an attested Trusted Execution Environment (TEE), so that out-of-scope queries are never evaluated and scope changes require fresh signatures from both parties. This protocol, which we call *Verifiably-Scoped Monitoring*, reconciles safety monitoring with zero-data-retention (ZDR): the company can verify that the lab's monitoring stays within the jointly-signed scope, while the lab still receives the monitoring outputs it needs.

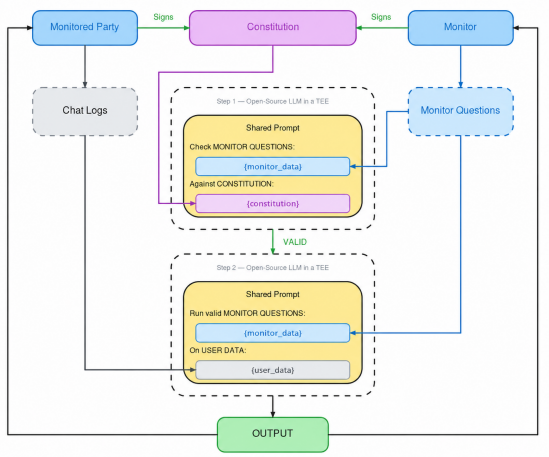

Figure 1. Verifiably-scoped monitoring applied to chat monitoring. The monitor M and monitored party P jointly sign a Plan π, whose *Constitution* defines the scope of permitted queries. At monitoring time, M submits encrypted queries while P submits encrypted chat logs to an attested TEE. An open-source LLM runs in two stages: it first checks each query against the Constitution, and only valid queries proceed to the second stage, where the classifier executes them against the logs. Outputs are released to both parties.

1. Introduction

There is increasing demand for monitoring AI-chat interactions in order to detect concerns such as misuse of dangerous capabilities or evidence of AI psychosis, and to collect information on the societal impacts of AI. The operational pressure to monitor is real and growing. However, the same infrastructure that detects things we broadly agree is harmful or collects aggregate statistics about patterns of use can just as easily be used to monitor market competition, employee sentiment, or political allegiance. A pipeline labeled "bioweapons misuse detector" and one labeled "political speech classifier" are, at the level of the underlying model, the same kind of system: a classifier reading conversations against a natural-language description of what to flag.

Today, the party being monitored has no technical way to distinguish these cases, and assurances about monitoring scope are contractual, not cryptographic: the monitored party is asked to trust that the monitor is doing only what it says it is doing. The result is a growing stalemate in which either (i) the monitor retains data and performs oversight, losing the customer's trust, or (ii) the monitor accepts zero-data-retention terms, and loses visibility into misuse.

Two recent efforts illustrate the gap. Anthropic's Clio (Tamkin et al., 2024) summarizes and clusters chat logs with an LLM pipeline that enforces privacy constraints; but Clio remains an internal Anthropic tool and users have no way to ensure that Clio's scope matches what Anthropic has documented publicly. OpenAI's "How People Use ChatGPT" analysis (Chatterji et al., 2025) verifies scope through human auditors (a committee of co-authors plus a secure data clean room), but again users cannot verify the claims and must trust the committee reviewing the analysis. Neither of these approaches lets a third party confirm, of an

[1]SPAR [2]Independent [3]Harvard University. Correspondence to: Benjamin Penchas <bpenchas@gmail.com>, Roy Rinberg <royrinberg@g.harvard.edu>.

Second Workshop on Technical AI Governance Research (TAIGR) @ ICML 2026, Seoul, South Korea. 2026. Copyright 2026 by the author(s).

arbitrary monitoring pipeline, that *this monitor is checking for exactly what it claims to check for, and nothing else*.

1.1. Contributions

This work is situated in the context of a larger body of work on the interplay of applying LLMs to the problem of secure multiparty computation. Existing work like Trusted Capable Model Environments (Shumailov et al., 2025) investigates the problem of using an LLM as a trusted third party, but does not commit to a concrete protocol. Attestable Audits (Schnabl et al., 2025) uses a TEE to verify a one-shot benchmark run against a fixed dataset, answering "did you run the benchmark you said you ran?"

Our work instead targets the related problem of enabling *verifiably-scoped monitoring*. We propose a two-party protocol (Figure 1) for a monitor and a monitored party to negotiate a Plan and use it to scope the monitor's queries.

Specifically, in this paper we:

1. **Formalize the problem** of scoped monitoring.
2. **Describe a potential solution** through a verifiably-scoped monitoring protocol.
3. **Discuss applications of this protocol**, including a scoped version of a zero-data-retention policy.

2. Related Work

Beyond the prior efforts named in Section 1, we position this protocol against five adjacent lines of work.

Trusted execution environments. A TEE is a hardware-isolated region of a CPU or accelerator that provides *confidentiality* (enclave memory is not readable by the host OS, hypervisor, or cloud provider) and *integrity* (the running binary is measured at launch, and any tampering is surfaced as an attestation mismatch). Modern examples include Intel SGX (Costan & Devadas, 2016), AMD SEV-SNP (Advanced Micro Devices, 2020), Arm TrustZone (Pinto & Santos, 2019), AWS Nitro Enclaves (Amazon Web Services, 2020), NVIDIA H100 confidential computing (NVIDIA Corporation, 2023), and Apple's Private Cloud Compute (Apple, 2024); remote attestation (Anati et al., 2013) lets a remote party verify that a specific binary is running inside.

Single-party LLM-based monitoring. The current state of the art runs monitoring as an internal lab function: Anthropic's Clio (Tamkin et al., 2024) clusters and summarizes chat traffic under privatized prompts, and OpenAI's usage analysis (Chatterji et al., 2025) classifies a random sample of conversations via a human-audited data clean room. In both, scope is a contractual property of a single party.

Multi-party trusted compute with LLMs. Recent work argues that an LLM inside a TEE can act as a programmable trusted third party, from systems (Shumailov et al., 2025) and economics (Stephenson et al., 2025) angles; neither commits to a concrete monitoring protocol.

Monitorability. Greenblatt et al. (2023) and Guan et al. (2025) ask the complementary question of whether a capable monitor exists at all; we take their answer as given and address the orthogonal governance question of how two parties agree on what the monitor may look for.

Aggregate privacy. Our protocol composes at the output stage with the k-anonymity, ℓ-diversity, and differential-privacy primitives (Sweeney, 2002; Machanavajjhala et al., 2007; Dwork, 2006).

3. Problem Statement

A **Monitored Party** P owns dataset D and wishes to permit monitoring of certain properties of D while keeping as much of the contents confidential as possible. A **Monitor** M wishes to evaluate specific properties about D through queries Q without revealing them.

These two parties agree on a **Plan** π, a natural-language agreement between P and M whose *Constitution* describes the scope of monitoring. The goal is to permit M to learn the outputs of Q_π applied to D, where Q_π is the subset of Q that adheres to the Plan π.

3.1. Desiderata

The actors may be adversarial: M may want to learn more than what P is willing to reveal. Since the queries themselves are not known to P, it is important that π correctly restricts the space of acceptable queries without destroying the system's utility for M.

D1. Data confidentiality. P learns only that conforming queries were executed and their results, not any of the contents of Q or Q_π. M does not learn anything about P's data outside of the agreed-upon scope.

D2. Execution correctness. The outputs represent a correct application of M's queries. Q_π consists of a faithful filter of Q through π, meaning queries are not incorrectly accepted or rejected.

D3. Scope confinement. M receives output only for queries that fall within the agreed Plan. Queries outside the Plan are never executed.

D4. Scope stability. The scope under which M may query P's data cannot be changed unilaterally. Any change requires fresh agreement from both parties.

The protocol of the next section satisfies these desiderata

by binding execution to a co-signed Plan inside an attested TEE: D1 and D2 follow from the TEE's confidentiality and integrity guarantees, and D3 and D4 follow from binding execution to the joint signature on π.

4. Protocol

We now describe a verifiably-scoped monitoring protocol addressing the problem statement and the desiderata. M and P negotiate a **Plan** π consisting of a set of actions, including conditions for determining Q_π. The Plan is executed by an open-source LLM classifier C running in a Trusted Execution Environment (TEE); the full pipeline is shown in Figure 1. The more general abstraction this protocol is a special case of is described in Section A.

4.1. Protocol Steps

First the parties negotiate a Plan, then each party sends its data. P's data is to be monitored and M's data is a list of monitor questions.

1. **Plan negotiation.** M and P codify the Plan π, whose *Constitution* (labeled in Figure 1) fixes the scope of actions permitted using M and P's data. Both parties sign a hash of the Plan, producing σ_P and σ_M, committing to conditions that are used to enforce scope stability.
2. **TEE setup.** The Plan is loaded into a *trusted execution environment* (TEE) that supports confidential computing and remote attestation, binding its launch state to the Plan and infrastructure code audited by both parties. It requires validity of σ_P and σ_M before proceeding.
3. **Data submission.** After verifying the attestation, M submits Q ("Monitor Questions" in Figure 1) and P submits D ("Chat Logs" in Figure 1), encrypted using the enclave key so both sets of data remain confidential to anyone outside of the enclave.
4. **Input filtering.** The first step of the Plan π is to validate both datasets and filter Q_π from Q.
5. **Query execution.** Each element of Q_π is executed by the classifier C, and the enclave releases the outputs to both P and M in the format fixed by the Plan π.

We apply this protocol to the deployment problem that motivates the work—reconciling enterprise zero-data-retention with safety monitoring—in Section B. Two further robustness concerns that any deployment must address are also included in the appendix: false positives in the input filter, handled by an appeals process (Section C); and prompt injection of the open-source classifier C, handled by mutual injection detectors (Section D).

5. Applications

Verifiably-scoped monitoring is, more broadly, a general primitive for verifiable commitments between parties that do not trust each other. Beyond chat monitoring it applies to many other settings such as narrowly-scoped review of a private codebase (an external reviewer asks a pre-agreed set of questions about source code without ever reading it) or auditing of redacted text data (an auditor confirms specific facts about a sensitive corpus without seeing the redacted contents). We highlight two applications we view as immediately implementable and impactful.

ZDR-lite. The first is a relaxation of zero-data retention we call *ZDR-lite* (described in more depth in Section B). Today an enterprise customer and an AI lab must choose between full retention, which gives the lab the logs it needs for safety monitoring but asks the customer to trust it with raw data, and full ZDR, which protects the customer but blinds the lab. ZDR-lite decouples *retention* from *access*: logs are kept only as ciphertext decryptable inside the co-signed enclave, and the lab's visibility is confined to queries within a jointly-signed Plan. This opens a point on the retention–monitoring trade-off that improves on both endpoints, and because it needs only one signed Plan and an attested classifier, we expect it to be tractable.

Recurring third-party audits. The second is recurring audits of an AI lab itself, in the spirit of the embedded evaluations some external organizations already perform. A verification scheme needs both a *positive* commitment, evidence that a claimed property holds, and a *negative* commitment, evidence that nothing outside the audit is happening. An auditor and a lab can co-sign a Plan encoding the auditor's questions (e.g., "was this personal data used in training?" or "was the model trained on its own chain-of-thought?"), run it as a monitoring pipeline inside the enclave over the lab's data, and release only a bounded output. The pipeline's output supplies the positive commitment; the negative commitment comes from the auditor having had access to the lab's whole system, including its data and part of its codebase, so it can confirm that no other process is being run unmonitored elsewhere. The critical ingredient is that the auditor returns periodically: on each visit it re-checks that the agreed pipelines are still wired into production and goes looking for new questions to turn into pipelines.

6. Discussion

The success of verifiably-scoped monitoring relies on the performance and accessibility of capable open-source LLMs and of TEEs. Recent progress on both fronts is encouraging: open-weight models continue to improve, and confidential-compute offerings on commodity hardware have moved from research demonstrations to production availability.

LLM Usage Statement

Large language models (LLMs) were used as general-purpose assistants throughout the preparation of this paper. Specifically, the authors used LLM-based tools for writing, revising, editing, and LaTeX formatting of the manuscript; for coding assistance in developing implementations of the protocol; and for generating figures. All such output was reviewed and edited by the authors, who take full responsibility for the final content, including all claims, citations, and the design of the protocol. We further note that open-source LLMs are also a core component of the system proposed here, where they serve as the in-enclave classifier.

References

Advanced Micro Devices. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more. White paper, AMD, 2020.

Amazon Web Services. AWS nitro enclaves user guide, 2020. https://docs.aws.amazon.com/enclaves/latest/user/.

Anati, I., Gueron, S., Johnson, S., and Scarlata, V. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

Apple. Private cloud compute: A new frontier for AI privacy in the cloud, 2024. Apple Security Research, https://security.apple.com/blog/private-cloud-compute/.

Chatterji, A., Cunningham, T., Deming, D. J., Hitzig, Z., Ong, C., Shan, C., and Wadman, K. How people use ChatGPT. Working Paper 34255, National Bureau of Economic Research, September 2025. URL https://www.nber.org/papers/w34255.

Costan, V. and Devadas, S. Intel SGX explained, 2016. URL https://eprint.iacr.org/2016/086. Cryptology ePrint Archive, Report 2016/086.

Dwork, C. Differential privacy. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 1–12, 2006.

Greenblatt, R., Shlegeris, B., Sachan, K., and Roger, F. AI control: Improving safety despite intentional subversion. *arXiv preprint arXiv:2312.06942*, 2023.

Guan, M. Y., Wang, M., Carroll, M., Dou, Z., Wei, A. Y., Williams, M., Arnav, B., Huizinga, J., Kivlichan, I., Glaese, M., Pachocki, J., and Baker, B. Monitoring monitorability. *arXiv preprint arXiv:2512.18311*, 2025.

Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. ℓ-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1 (1):3–es, 2007.

NVIDIA Corporation. NVIDIA H100 confidential computing, 2023. Technical brief, NVIDIA.

Pinto, S. and Santos, N. Demystifying Arm TrustZone: A comprehensive survey. *ACM Computing Surveys*, 51(6): 1–36, 2019.

Schnabl, C., Hugenroth, D., Marino, B., and Beresford, A. R. Attestable audits: Verifiable ai safety benchmarks using trusted execution environments, 2025. URL https://arxiv.org/abs/2506.23706.

Shumailov, I., Ramage, D., Meiklejohn, S., Kairouz, P., Hartmann, F., Balle, B., and Bagdasarian, E. Trusted machine learning models unlock private inference for problems currently infeasible with cryptography, 2025. URL https://arxiv.org/abs/2501.08970.

Stephenson, M., Miller, A., Sun, X., Annem, B., and Parikh, R. NDAI: Non-disclosure AI agreements. *arXiv preprint arXiv:2502.07924*, 2025.

Sweeney, L. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

Tamkin, A., McCain, M., Handa, K., Durmus, E., Lovitt, L., Rathi, A., Huang, S., Mountfield, A., Hong, J., Ritchie, S., Stern, M., Clarke, B., Goldberg, L., Sumers, T. R., Mueller, J., McEachen, W., Mitchell, W., Carter, S., Clark, J., Kaplan, J., and Ganguli, D. Clio: Privacy-preserving insights into real-world AI use, 2024. URL https://arxiv.org/abs/2412.13678.

A. General Two-Party Protocol

The protocol of Section 4 is one instantiation of a more general two-party pattern which we illustrate in Figure 2. In the general form, two parties A and B jointly sign a natural-language Plan π (a "Shared Prompt" in Figure 2) that fixes the scope of a joint computation over their respective inputs; the Plan is then executed by an open-source program inside an attested TEE. Each party submits its encrypted data using the enclave's public key, the program runs the steps fixed by π, and the enclave releases only the outputs π specifies to the recipients π specifies. Neither party observes the other's data; neither party can change π unilaterally; no computation outside the scope of π is performed.

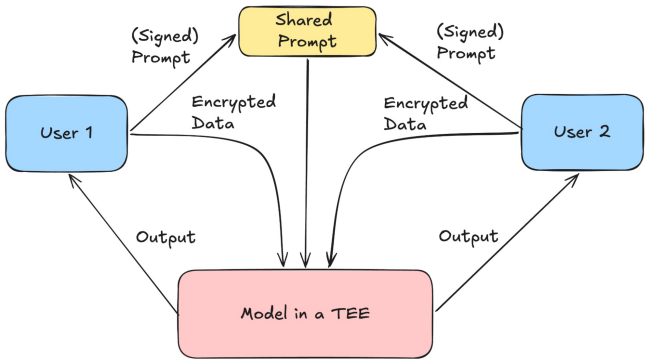

Figure 2. The general two-party protocol.

B. Reconciling Safety Monitoring with ZDR

We expand here on *ZDR-lite*, the relaxation of zero-data retention introduced in Section 5, which reconciles safety monitoring with the zero-data-retention contracts now common in enterprise AI deployments.

The problem. Zero-data-retention (ZDR) is an increasingly common contractual requirement for enterprise API customers. Companies handling sensitive data (legal, financial, healthcare, defense) want the AI lab to retain no conversation data after inference. AI labs, however, are reluctant to offer ZDR: retention is what makes safety-relevant monitoring possible at all. Without logs, the lab cannot aggregate usage signals for abuse detection, flag patterns of jailbreaking or prompt injection across a customer's usage, or surface evidence of autonomous scheming. In practice, ZDR is offered only to organizations the lab already trusts considerably.

A verifiably-scoped ZDR policy. Verifiably-scoped monitoring lets both parties get most of what they want by decoupling *retention* from *access*. The scheme is illustrated in Figure 3.

1. The enterprise customer (EC) and the AI lab agree on a *shared Plan* π that defines the scope of monitoring the lab is permitted to perform: which categories of query are allowed, what output format is required, any aggregation floor, etc. The Plan fixes the scope of monitoring, but not the specific queries themselves. Both parties sign π.
2. The attested TEE generates an encryption keypair inside the enclave at first boot and publishes the public key `TEEPubKey` alongside its attestation, which binds `TEEPubKey` to the specific co-signed binary measurement. The corresponding private key `TEEPrivKey` never leaves the enclave in plaintext; it is *sealed*[1] to disk so that only a fresh boot of the same attested image on the same physical chip can recover it.
3. As chat logs are produced, the lab encrypts each log under `TEEPubKey` and stores the ciphertext in a per-customer database. The lab never holds the decryption key, cannot read the logs, and the database holds only ciphertext.
4. When monitoring is run, three things enter the attested TEE: the signed Plan π (via the co-signed artifact); the ciphertext logs (pulled from the per-customer database); and the lab's queries $\{q_i\}$ (sent directly by the lab to the enclave, and not visible to EC).
5. Inside the enclave, the verifiably-scoped monitoring protocol of Section 4 runs: the TEE decrypts logs using its sealed `TEEPrivKey`, checks each q_i against the scope defined by π, applies C on (q_i, log) under the constraints of π, and returns only the resulting classification labels to the lab. No plaintext log ever leaves the enclave, and any q_i outside π's scope is rejected.

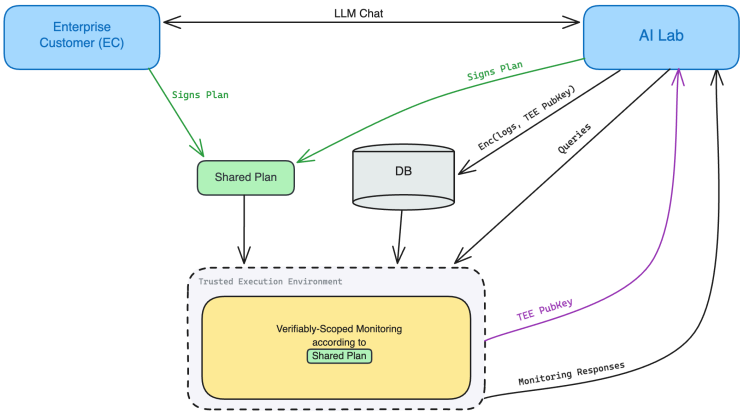

Figure 3. A verifiably-scoped ZDR deployment. The attested TEE publishes an enclave-bound public key alongside its attestation; chat logs are encrypted under that key by the lab and held in a per-customer database. Both parties sign a shared Plan; the TEE pulls the ciphertext logs from the database, decrypts them inside the enclave using its sealed private key, and runs the verifiably-scoped monitoring protocol of Section 4 against the co-signed Plan, returning only the agreed monitoring responses to the lab.

What each party gets. Under this scheme, the AI lab does not store any plaintext logs and cannot directly access them; its only visibility into customer traffic is through queries that adhere to the Plan it and the customer have jointly signed, executed by an attested binary it cannot silently modify. The enterprise customer, in turn, gives up full ZDR in a very narrow sense (logs are retained as ciphertext decryptable only inside the co-signed enclave) but has the ability to limit what the lab can ask through Plan negotiation: any query outside the shared Plan is rejected by the TEE, and any change of scope requires the customer's fresh signature. We stress that ZDR-lite is no longer zero-data retention in the strict sense: the (encrypted) logs are retained rather than deleted, and a lapse in the enclave's security could in principle expose them. What the customer gains in exchange is a cryptographic guarantee that the logs are decryptable only inside the co-signed enclave and are read only by queries within the signed scope.

C. An Appeals Process for False Positives

The ZDR deployment of Section B is one application of the protocol; the appeals process described here is an additional feature, common to any deployment, that handles what happens when the classifier is wrong. A false positive in the auditor occurs when the TEE incorrectly judges a query to be out of scope of π. We propose a structured appeals process around such flags for two reasons. First, a confirmed scope violation may trigger a costly downstream process (review, suspension, escalation), so the rate of unjustified flags should be kept low. Second, a working appeals path lets us tune the classifier to flag more aggressively, accepting more false positives in exchange for fewer false negatives, without that tradeoff falling unilaterally on the monitored party.

We propose that any positive flag triggers an appeals window (Figure 4) in which P may attempt to provide *exonerating evidence* of one of two forms:

1. **Small modification.** A small modification to the data that removes the flag (e.g., a typo in the underlying data caused the classifier to trigger; correcting the typo causes the classifier to no longer trigger).
2. **Verifiable secondary query.** A reasonable explanation for the flag that can be verified by the TEE using a different prompt π' (e.g., "ask the TEE this other question, and the answer will demonstrate that the flag was a false positive").

The appeals process is itself bounded by the same TEE-enforced guarantees as the original protocol (both forms of exonerating evidence are evaluated inside the TEE under signed prompts), so the appeal cannot be used by P to exfiltrate data, nor by M to broaden the effective scope of monitoring.

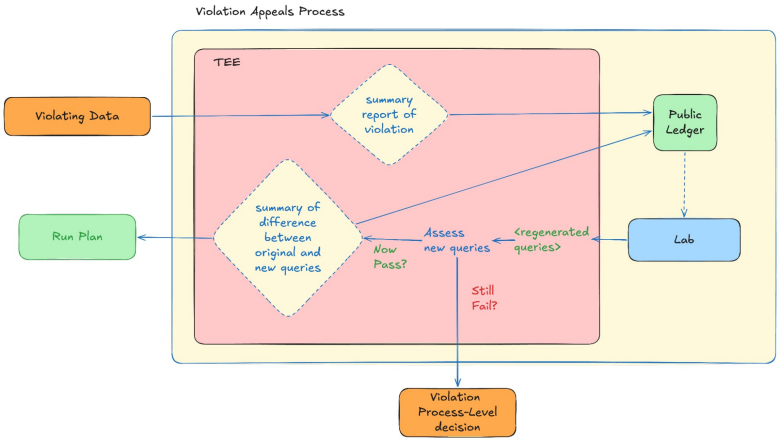

Figure 4. Sketch of the violation appeals workflow. The monitored party can supply either a minimal data modification or a verifiable secondary query.

When an appeal succeeds, the TEE writes a short record to a per-Plan append-only public ledger: an `appeal_accepted` event together with a 5–12 word semantic-diff summary of the change emitted by a LLM inside the enclave (e.g., "Corrected a misspelled name."). The ledger likewise records all `query_rejected`, `appeal_filed`, and `appeal_rejected` events. No raw data, query text, or step-output content ever appears in it. This gives both parties an external, tamper-evident record of how the appeals path is actually being used.

D. Out-of-Distribution Inputs and Prompt Injection

Because the inputs submitted to the enclave are private, there is always a risk that the classifier C behaves unexpectedly on an input far out of its training distribution. The adversarial version of this is a *prompt injection*, in which a party embeds instructions in its input that manipulate C's output: the monitor could embed instructions in its queries that cause C to suppress flags it would otherwise raise, and the monitored party could embed instructions in its data to the same end.

This risk is sharpened by a property our protocol otherwise relies on: C is open-source, so that both parties can inspect what runs. Prompt injection typically requires trial-and-error and rarely succeeds on the first attempt, but an open-source C lets an adversary probe a local copy until it finds an injection it is confident will transfer. We propose two complementary defenses.

Do not commit to a single model. An input that is out-of-distribution for one model is usually in-distribution for another. The Plan can therefore authorize several interchangeable models and select one at runtime using randomness drawn inside the enclave. A successful injection must then defeat *all* of the candidate models at once, which is substantially harder than attacking a single known target.

Opaque mutual injection detectors. Each party may interpose its own black-box prompt-injection detector on the other party's input before it reaches C (Figure 5). Each detector is opaque to the other party: neither side can inspect the other's. The detectors need not be highly accurate; because the adversary cannot determine which inputs will trip a detector it cannot probe, it cannot inject with confidence even against a detector with a modest detection rate. Each party need only trust its own detector.

A detector that fires can itself produce a false positive; the flagged party contests it through the same appeals process used for any other false positive (Section C), while a confirmed injection attempt is handled out-of-band through the parties' organizational or contractual relationship. Together these yield a *panopticon*-style argument:[2] it is strongest when the penalty for a detected attempt is large relative to the per-attempt gain, and weakest when an attacker can make many cheap attempts.

Notes

[1]The TEE hardware exposes a sealing key derived deterministically from the running binary's measurement and a per-chip secret fused at manufacture. The enclave encrypts `TEEPrivKey` under this sealing key and stores the ciphertext as an ordinary file. On the next

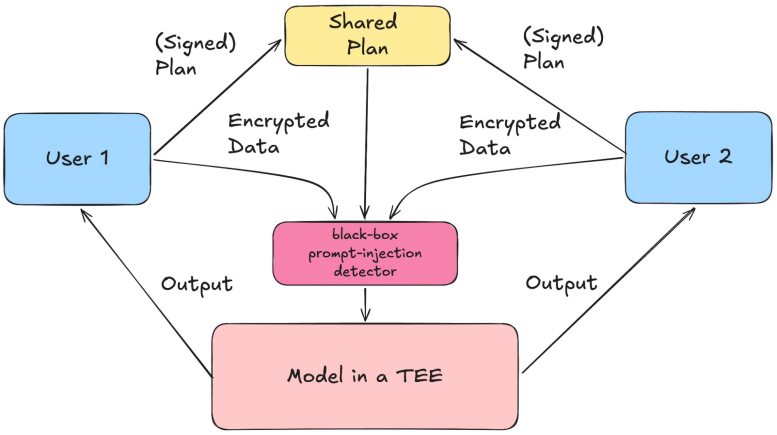

Figure 5. Black-box prompt-injection detector interposed between the signed Plan, encrypted data, and the TEE-hosted model C. Each party's detector is opaque to the other, so neither party can reliably craft injections that evade detection.

boot, only the same attested image running on the same physical chip can re-derive the sealing key and recover `TEEPrivKey`; any other binary, any other chip, or any read by the host gets only opaque ciphertext.

[2]In a panopticon, a subject cannot tell whether they are being observed and so behaves as if they always are. Here, because a detected injection attempt is costly relative to its expected payoff, the rational move is not to attempt one.