Working Paper

A System Overview for Near-Term, Low-Trust AI Compute Verification

Naci Cankaya
naci@intelligence.org
Machine Intelligence Research Institute
Technical Governance Team

This is a working draft of my current best idea for a privacy-preserving, retrofittable AI compute verification system, for confidence-building in an arms-control-like AI agreement between rival nation states. The purpose of this draft is to elicit community engagement by making use of Cunningham's law: I make assertions about what the (emerging) field of AI verification should aim for, and people with experience in international policy, cybersecurity and any relevant field of engineering can point out what this draft gets wrong. Throughout this draft, I added questions aimed at domain-experts. It is easiest to comment on the LessWrong version of this paper.

Thank you to everyone who has provided feedback to version 0.1.

1. Introduction and summary

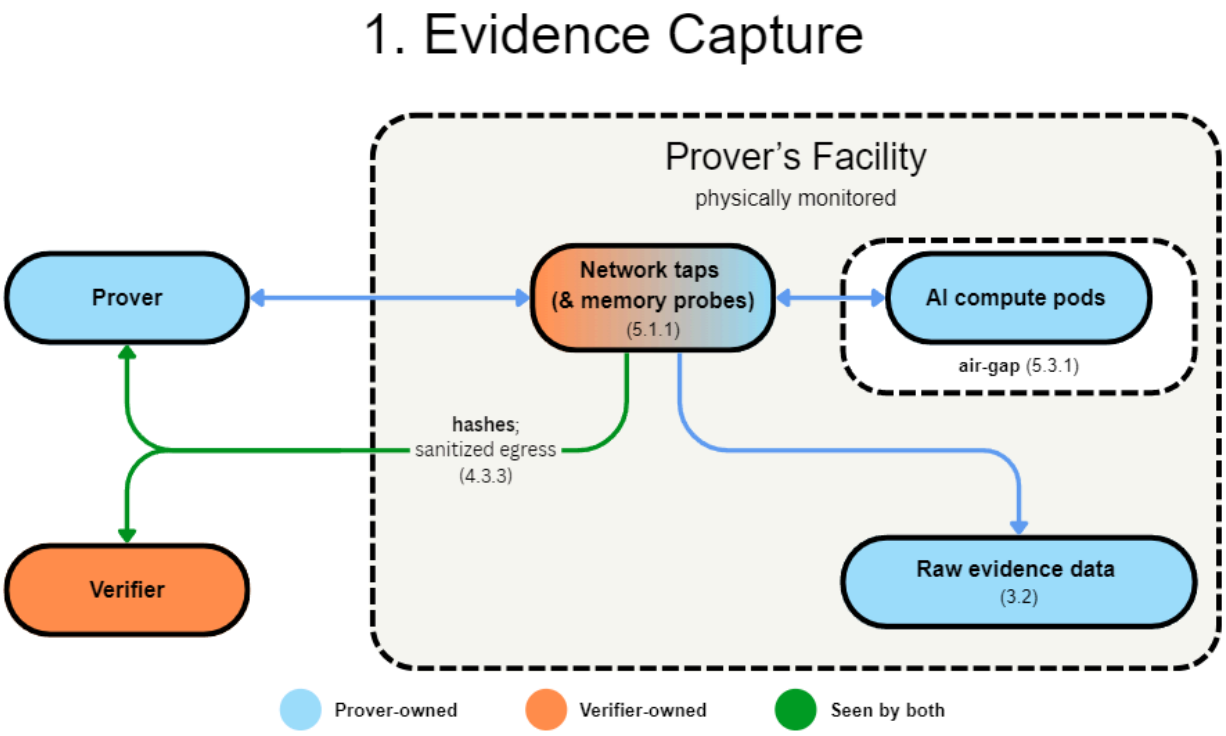
In order to plan and execute under tight timelines, one needs to make some strategic bets, instead of hedging too much and keeping all options open. The field of research on AI verification is bottlenecked partly on lack of shared vision (as well as human capital, but having clear goals helps hiring and fundraising). With this post, I aim to:

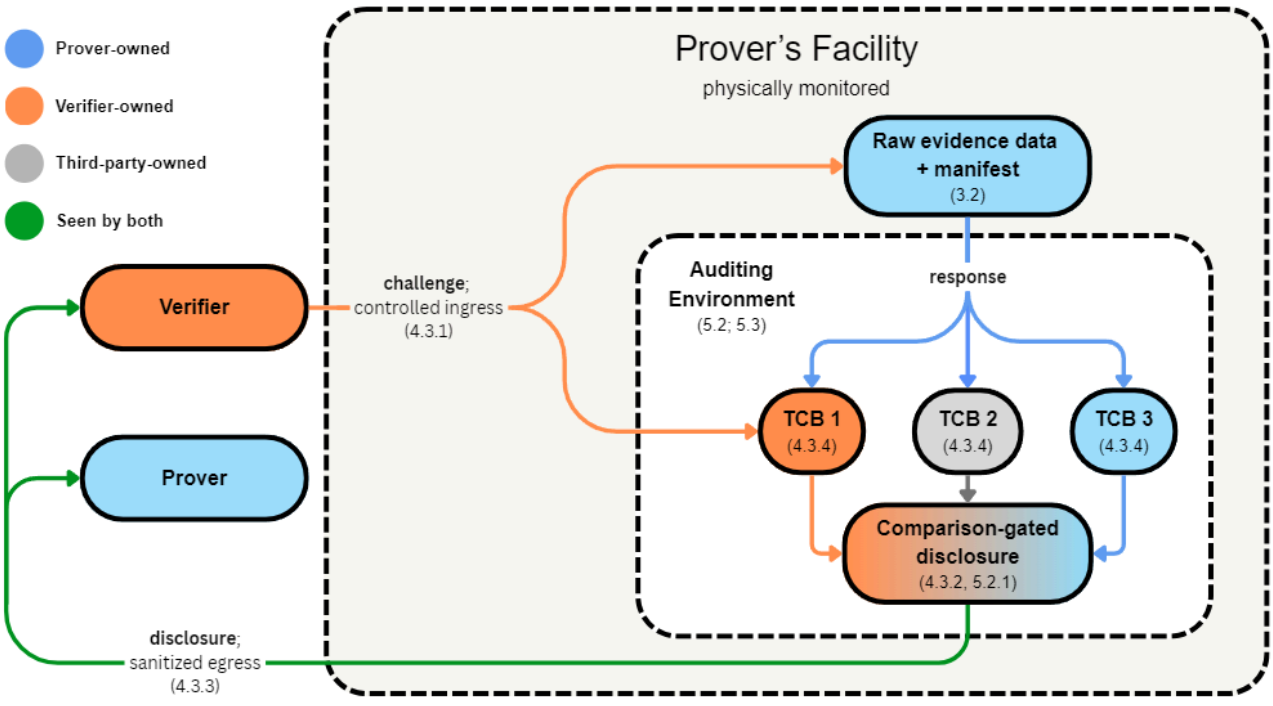
1. Make technical objectives for verification in high-stakes AI governance more specific and actionable (section 2)
2. Contribute a first, high-level reference architecture for meeting these goals. (section 3 and 4)
3. Gather an overview of relevant work in the field, both prior and ongoing.

You, the reader, may disagree with some of the items in my problem statement, or the technical approach of the reference architecture. You may find some points too under-specified to even be directionally useful. If you do, please share your insights as comments and explain your reasoning. The reference architecture will (hopefully) improve through red-teaming on the drawing board, before it is ready for red-teaming in the lab.

The most important properties of the proposed system:

1. **Privacy of AI users and IP is preserved** and only pre-agreed information is disclosed. Confidential data does not leave the monitored facility, not even in encrypted form.
2. **The system is retrofittable** to existing AI hardware. It does not require fabrication of newly designed technology, and can (in principle) be built from off-the-shelf components.
3. **Physical security is a prerequisite**: Access controls, guarding and inspection need to ensure that devices are wired and installed correctly, but they do not need to perform deep inspection of silicon. PUFs, tamper evident seals, hardware signing keys etc. are not relied on as the only layer of defense against counterfeits and tampering.
4. **There is no need for *mutually* trusted chips**, and the surface area for attacks against *unilaterally* load-bearing logic is reduced aggressively, by separating performance-heavy hardware (e.g. for re-execution) from minimalist, security-critical trusted computing bases (TCBs; *unilaterally* trusted; e.g. for hashing, comparing and information-gating).
5. **Evidence capture and commitment is separate from evidence evaluation.**
    a. **Network taps** are the default mechanism for evidence capture, with optional additions such as **memory pinging**.
    b. Evidence evaluation happens in **air-gapped environments** (tentative plan A), or optionally via **zero-knowledge proofs** (tentative plan B, if and when they mature and become a better choice). Both approaches use random sampling from pre-committed records, to keep verification overhead low.
    c. **The system is a versatile platform for a wide variety of verification goals:** Inference vs. training, deployment of only whitelisted models (and scaffolds) for non-blacklisted uses, etc. The captured evidence can identify each forward pass uniquely, and attribute it to the hardware and time it was processed on.
6. **Additional mechanisms defend against evasion or leakage:** Air-gapping and physical monitoring; redundant re-computation on multiple, diverse TCBs; resource accounting; memory wiping.





On a high level, the proposed system works as follows:

Network taps (and other optional mechanisms) capture evidence of what workloads ran on the prover's hardware. This information is confidential, as it contains business secrets and customer

data. For this reason, the verifier merely receives cryptographic commitments of this evidence (signed hashes). The prover stores the plaintext evidence for as long as needed. Given these commitments, the verifier can retroactively challenge the prover to enter (a random subset of) the plaintext evidence into a secure evaluation environment, one that the verifier relies on to accurately generate a disclosure in a pre-agreed format, and the prover relies on to not reveal any further information. **Apart from inspectable analog components, no digital part of the system needs to be *mutually* trusted. The security of both the network taps and the evaluation environment rest on redundant re-computation and cross-checks** (plan A, superseded by Zero-Knowledge-Proofs if and when they become viable)**, and physical monitoring.**

- In section 2, I motivate the design choices of this system.
- In section 3, I discuss the bird's eye view of the architecture, and an end-end trace through the system, from the workload running on the prover's ML servers, all the way to the disclosure produced by the auditing environment.
- In section 4, I discuss the sub-systems that control information movement, in order to eliminate the need for mutually trusted chips.
- In section 5 we move one abstraction layer lower, down to the individual components: Network taps, software, analysis methods and defense mechanisms against evasion or leakage.

2. Problem statement and motivation

Here I provide motivation for the reference architecture and its specific design choices.

2a. Low-trust AI governance

Suppose the U.S. government, China, and other jurisdictions were to seek an agreement for more responsible AI development and deployment. Here is a demonstrative list of rules that these parties may agree on:

1. **Maintain monitorable chains of thought, refrain from developing frontier AI with neuralese architectures**, or putting it in charge of high-stakes decisions/giving it tool-use interactivity with the world.[1]
2. **Require AI systems (models and scaffolding) to be reported during and after training.** All checkpoints (and their intermediate training data) ought to be cryptographically committed (not revealed) to at least one independent governing body. Same goes for model scaffolds and everything else used for elicitation & inference.
3. **Require AI systems to be whitelisted before deployment, including internal-only deployment.** A criterion for whitelisting may be for an AI system to pass a standardised

[1] Monitorable means CoT without which an AI can not perform its tasks, and CoT that accurately reflects an AI's goals and actions, in a way its human supervisors can understand in quasi-real time. Also maintain monitorable inter-agent communication.

suite of evaluations, which could be updated as evals advance and requirements change.

4. **Create and enforce a blacklist of illicit use cases.** Such a blacklist may include covert AI-enabled influence/deception campaigns, fraud/ransom, or aiding non-whitelisted users in high-risk dual-use areas such as CBRN.
5. **Annual or bi-annual summits for AI security/safety.** Leaders in AI research and governance gather to share safety research results and update their consensus on risks, evaluations and standards.

Whatever rules actually do end up on the negotiating table between China, the US. etc. will—to a large part—depend on their verifiability. In our example, rules 1 to 4 may already warrant *adversarially robust* verification, especially if there are no exemptions for military AI systems and if there is a serious concern that covert evasion could give a cheater a strategic advantage. If slowdown or other restraint of AI development is part of the agreement, verification is indispensable.

I expect that verification of only high-level aggregate properties such as A) total training FLOPs of a model or B) fraction of training vs. inference compute is not sufficient against covert AI development or deployment. Such coarse properties provide little information about compliance with rules such as 1-4.
For this reason, I assert *collision-resistant evidence capture of individual ML workloads* as a requirement for the verification system, as well as support mechanisms against hidden workloads.

Questions for the reader #1

The requirement of capturing evidence that can uniquely identify each ML workload (as in, every forward pass) is what I settled on. The reasoning behind this is that with this evidence, you can —in principle— screen for any governance objective. Do you think this is trying too much? What verification goals do you think the field should actually focus on?

2b. Threat model

We distinguish between a prover and a verifier. In the most challenging case (which we assume for generality), the two are nation-state adversaries with the most advanced cyberoffense capabilities in the world.

1. **The prover** is the party that owns and operates the hardware under scrutiny. In the worst case, which I assume, they are a colluding pact of a government and all its domestic companies. A malicious prover aims to covertly break the agreement's rules. The most challenging violation to defend against is the prover covertly running whitelisted models for blacklisted uses. If the verification system can detect such violations, it can also detect inference of non-whitelisted models, or covert training.[2]

[2] Assuming the verification is based on re-execution in order to explain network traffic (and memory images).

2. **The verifier** is the party demanding evidence for the prover's claims, and who insists on independent verification of this evidence. In the worst case, which I assume, they are also a colluding pact of a government and its domestic companies, aiming to exfiltrate the prover's secrets (weights, code, training data etc.).

The ideal for a verification system is to provide sufficient assurance to the verifier that evasion of the installed system is either very likely to be caught, or too small to be consequential. At the same time, the presence of this system should not lower the security standard of the monitored facility.

Covert adversary and the inversion of the fortress problem

Given the cyberoffense capabilities of (AI-augmented) nation-state adversaries, intuition would suggest that perfect security is required everywhere, i.e. that not a single covert ML workload should be physically possible.

However, this is not an adequate threat model in this scenario. Even in the worst case, members of an agreement wish to stay covert in any attempted evasion, or attempted attack against a verification system. In "Security Against Covert Adversaries", Aumann and Lindell describe the **covert adversary threat model**: cheating is deterred when detection likelihood × cost of getting caught exceeds expected gain. With proper implementation, this inverts the strategic calculus: An openly malicious adversary can attack as often as they want, with any available method. But a covert adversary is caught if they fail to stay hidden even once. This allows the defender to layer multiple detection mechanisms until cheating attempts are no longer worth the risk. **A covert adversary is (far) less costly to defend against, since random sampling only needs to catch a single instance of cheating.**

The attribution problem and plausible deniability

In reality, a technical mismatch (e.g. a hash failing to verify) does not come with a label indicating whether it was caused by a covert evasion attempt, a cosmic-ray bit flip, or evaluation software misjudging an edge case. If a covert adversary knows that detected anomalies can be plausibly waved off as a malfunction, the deterrence becomes less effective. If the verifier treats every technical bug as a deliberate treaty violation, the agreement will quickly collapse under false positives. (Past geopolitical intelligence failures, such as the overconfident assessment of WMDs in Iraq, highlight the danger of escalating based on ambiguous technical indicators).

For the verification protocol to work, the system must undergo testing to identify typical false positives and their origin. For deployment, the prover and verifier must agree on a procedure for escalating levels of disclosure and investigation, one that robustly terminates in attribution to either technical error, or malice.

Assumptions about physical security and inspection

These are assertions based on my research experience so far, and conversations I have had with security experts. Please let me know if you, the reader, disagree with any of these statements.

Monitored continuously:

- Who enters or leaves a datacenter, and what devices they interact with while inside (for inspection or maintenance purposes)
- Whether the prover's servers emit suspicious amounts of RF or acoustic signals. This is measured by verifier-installed sensors. These sensors do not send signals to the verifier, other than an alarm in case of a violation.
- The cabling of network taps and other sensors installed by the verifier
- The outside perimeter around the facility

Inspected occasionally (random samples):

- All load-bearing analog components, including data movement control systems discussed in section 4.2 and 4.3
- Whether the verifier-installed analog sensors behave as specified (checked by both the prover and the verifier). "Anaolog sensors" refers to sensors used to monitor air-gaps, not compute.
- Whether a device (taps, or any computing base in an auditing environment) was made by the prover, verifier, or someone else
- Any anti-tamper seals, if present

Always untrusted, even if inspected:

- Any semiconductor-based information processing device not domestically fabricated and assembled by the relying party (see section 4.1)
- Any software that is neither formally verifiable, nor made domestically by the relying party

Discussion of attack surfaces

What complicates this assumption is that attacks can be tested in secret, on replicas of the verification setup. An attack on a real, deployed system would only be attempted for those attacks that have proven reliable enough in the tests. However, this is still a higher bar, since the attacker would need to penetrate all layers of defense and redundancy simultaneously, at every attempt.

Against a correctly installed setup, they would need to find an exploit that open red-teaming efforts have not found, against a system whose security-critical components are either analog, software-free FPGA appliances (the network taps) or very minimal TCBs made by an adversary.

Instead, an attacker may attempt to attack supply chains, the construction process, or the deployed hardware by physically accessing and replacing/modifying hardware. To defend

against this, physical security and randomly sampled inspections are likely to be the primary defenses.[3] I do not go into detail in this version, but note that a reference architecture needs to specify not only the deployed system, but also how to secure each step of the lifecycle.

A malicious prover faces the challenge of needing to not only shut down the verifier's monitoring hardware (and optionally hardware of a third party as well, at the same time), but they also need to make this hardware forge plausible outputs (that will hold up in future ZKPs). Otherwise, the verifier would notice an absence of hash commitments from their monitors, or missing disclosures from audits.

A malicious verifier aiming to exfiltrate confidential information would need to penetrate multiple layers of defense. The question here is whether or not the retrofitted system introduces an attack surface. If the verifier's hardware was malicious, it would need to either A) exfiltrate information through either the same fiber link that is split and fed to the prover (see section 4.3.3), or B) through side-channels that somehow breach the prover's facility boundary without being caught, or C) by compromising inspectors and guards. Against C), the prover may require that any team of inspectors is made of people with diverse backgrounds, keeping watch on each other.

Questions for the reader #2

The threat model section is under-developed and would benefit most from input from cybersecurity experts, and people generally experience with AI threat modeling.

What scale of covert evasion would be consequential, to what degree? E.g. does defeating the slowdown goal of a treaty require multiple covert training runs, and at what scale? What about inference and RL?

What makes hardware unilaterally secure (in the covert adversary threat model) against attacks such as rowhammer, fault injection, spectre etc, provided that abnormal behavior (like the repeated access patterns in rowhammer) can be actively monitored against?

How to deal with the attribution problem in a way that satisfies both parties? What pre-agreed procedure would draw from lessons in CBRN verification, but translate well to AI compute verification?

2c. Practical requirements

The space of verification options is very broad, so to narrow it down, I have made some assumptions about what practical requirements a verification system should satisfy.

[3] If the verifier compromises a network tap, for example, they would still need a covert communication channel to exfiltrate secret data. Since all information egress of the tap is cloned to the prover, even a fully successful attack against the network taps does not defeat the system: The prover would catch information exfiltration via re-computation of hashes, and physical side-channels are erased by the OEO gateway (see section 4.3.3).

1. Retrofittable to existing AI hardware
One requirement is the time to deploy. This motivates retrofitability: The verification mechanisms should not rely on AI hardware having trustworthy monitors installed off-the-shelf. Apart from the lead times of AI hardware being measured in years, there is also the unsolved challenge of proving the absence of supply-chain attacks in advanced silicon. I comment on this in section 4.1.

2. Low-friction integration
The retrofit should be substantially cheaper than the monitored hardware. I expect that political and commercial acceptance will depend on the cost being below 10%, ideally below 1%. Of course, in order to avoid catastrophic AI risks such as loss of control, it would be sensible to pay huge costs, but the cheaper verification can be made the better.

The verification system should not introduce a security liability that adds significant risk of exposure of the prover's data. The security of this system should be at least up to the same standard as the security of the monitored AI datacenter, with the added requirement of being secure against *both* the owner of the datacenter *and* the verifier the system responds to.

Finally, the verification hardware should not rely on new, yet unvalidated technology. This flows from the time-to-deploy requirement, which should be measured in weeks or months, not years. Supply chains should already exist, with the capacity to ramp up quickly enough.

3. Automated and scalable
Manual inspection should be eliminated where possible, for both confidentiality and scalability. Millions of workloads will be screened daily, even with sparse random sampling.

4. Flexible and versatile for changing requirements
As rules change, the verification system should be able to adapt rapidly. This means that the captured evidence should be informative enough to be screened for a broad range of possible criteria, and that the compliance screening should be programmable without requiring new hardware.

I expect token/symbol-level input-output data of ML workloads to be an adequate choice, as it is the most semantically meaningful compared to alternatives. To provide further collision-resistance (against slightly different ML workloads with the same output tokens), one could require additional evidence such as projections or hashes of intermediate activation tensors. This becomes even more robust if the activation tensors are mapped in real-time using verifier-provided nonces or tensors.

5. No single chain of trust
Hardware-based security and attestation mechanisms rely on trust in the supply chain. A prominent example for this are Trusted Execution Environments (TEEs). In an adversarial international setting, it is unlikely that commercial hardware and software vendors will be trusted

to successfully defend against foreign governments, let alone their own governments. Therefore, the verification system should leverage redundancy.

Ideally, non-programmable monitors such as network taps would be built from multiple and separate devices, each one being as minimalistic as possible (small TCB) and sourced from different supply chains. Outputs of network taps (hashes) could then be computed redundantly and cross-checked by these processors, a process which I detail in the “Network taps” section.

The programmable part(s) of the system (i.e. the evaluation environment) will inevitably require larger TCBs with more attack surface. However, redundancy, air-gapping and minimalistic disclosure are layers of defense whose implementation I will detail in the “Secure Evaluation Environments” section.

6. In the foreseeable future, physical monitoring against hardware tampering is necessary.
Making tampering of hardware evident without physically guarding said hardware is an ongoing field of research, and mechanisms such as tamper-evident seals and cameras, or physically unclonable functions (PUFs) are a helpful layer of defense. However, full anti-tamper assurance against a highly capable adversary, without full knowledge of their capabilities, is unlikely to be solved in the near future (if ever). Therefore, I assume that the verification system is physically monitored while deployed; by cameras, sensors or even vetted guards of diverse backgrounds.

Questions for the reader #3

Do you agree with these requirements, and the reasoning behind them?

3. System overview and operation

3.1 Brief introduction

One key feature of the proposed system is the separation of evidence capture and evaluation. The general principle is “capture everything, challenge and audit random samples later”.

Non-programmable, minimalistic network taps
Network taps are placed inside the prover’s datacenter, and are therefore subject to strict physical security requirements. Beyond hashes, no information is to be shared with the verifier. At the same time, the devices ought to be resistant against software attacks of any kind. A device with practically zero attack surface, easily verifiable correctness, and a tightly controlled output is the only kind that meets all three requirements simultaneously. This is much more tractable if the logic of network taps is both minimal and non-programmable. One suitable choice (at the same level of complexity as active wardens, which the taps use anyway, see section 5.1.1) would be fixed-function logic parsing only L2-L4 structure, never interpreting L7

payloads. Re-assembly of meaningful objects happens inside the evaluation environment. In such a setup, the bitstream would either need to be fully reproducible[4] or stored[5].

Regardless of whether challenged hashes would be satisfied by either retrieval from storage or reproduction of network bitstreams, the committed evidence is fully opaque about not just the content, but also the *content type*. Hashes do not individually map to semantically meaningful objects (e.g. files), and such objects would be sliced arbitrarily. Without any further supporting mechanisms, this would mean that the verifier can at best only challenge intervals of commitments, and choose large enough intervals such that they contain complete files that the evaluation environment can work with.

Beyond this minimalist version of the reference architecture, the verifier may require commitments to cleanly map to whole objects. The prover can offer untrusted, but non-load-bearing[6] labels: A **manifest** of what hash intervals map to what objects. Example: "The preimage network traffic bytes of all hashes from timestamp A to timestamp B contain JSON file #j294875229, which is a prompt completion for request #r294875228."

This manifest can be understood as the prover's claims about what the ML servers were used for at any given time, and how the captured network traffic maps to these claims.

Questions for the reader #4

Does it make sense to keep the tap logic as barebones as this? Or can more functionality such as object awareness be integrated while not introducing an attack surface that is more significant than weaknesses you see elsewhere in the system? How important is it to enable the verifier to know what kinds of objects they are challenging?

Confidentiality and security

The verification system does not transmit sensitive data (network traffic and memory images) of server hosts or cloud tenants out of the monitored datacenter. Only cryptographic commitments (hashes) are sent to a verifier outside of the facility. When the verifier challenges a subset of these commitments, the preimage data enters an auditing environment *inside* the prover's facility.

Programmable, retroactive compliance screening

Evidence evaluation can and should be retroactive. Verification of semantics (e.g. the intended purpose of an ML workload) can be hard to define programmatically. At this abstraction level, compliance screening is inherently complex. Moreover, any single analysis heuristic may be

[4] Making network-packet generation bitwise reproducible requires bypassing the operating system's TCP/IP stack, whose packetization decisions vary across runs. A custom deterministic networking stack can emit bitwise-reproducible Ethernet frames, demonstrated across multi-node TCP deployments in soon-to-be-published research.

[5] Bitstream storage and payload re-assembly is a solved engineering problem, see the commercial precedents listed at the top of 5.1.1.

[6] The evaluation environment can confirm claimed labels. The prover providing labels to the verifier only serves the purpose of enabling the verifier to make more meaningful, object-type targeted challenges.

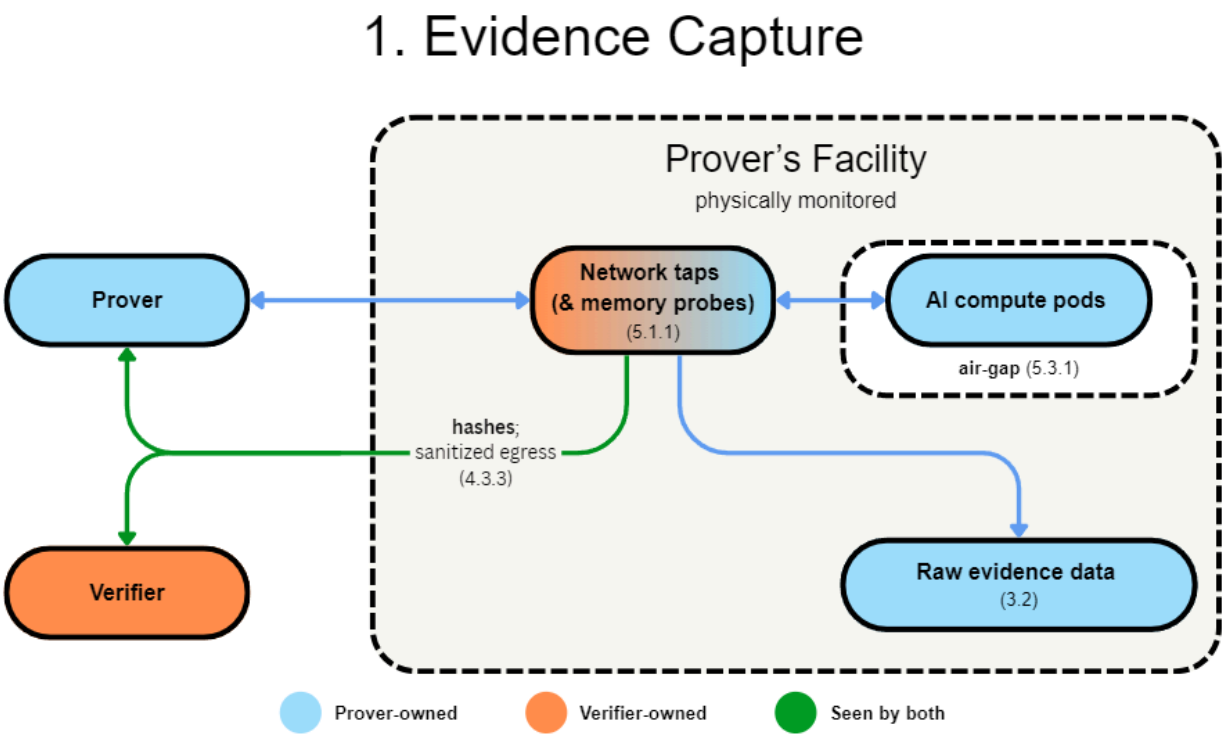
gameable, but if historical records can be re-visited with newer software, the attacker faces the problem of needing to anticipate what patterns or tricks future software would miss. For this to work, the auditing environment must be programmable and capable of complex computation (e.g. to run inspector agents, see section 5.2.3).

3.2 End-end execution trace

Premise. The trace assumes the system is live and operational, and I treat a spec for secure setup as out of scope. One more assumption: The prover cannot communicate with a monitored pod except across a tapped and warden-sanitized frontend network link.[7] In principle, all (or almost all) egress traffic out of a monitored set of ML servers should be fully explainable (that is, independently re-computeable) by the observed activity of the system.

We follow one inference request, but the tap sees it only as part of an undifferentiated Byte stream. The request is never singled out *at capture*, only *reconstructed later* in the auditing environment.

3.2.1 Evidence capture



[7] Backend traffic (compute and storage fabrics) are much higher-bandwidth, and therefore more expensive and challenging to tap into. They are also latency sensitive, meaning that active wardens at such links would hurt performance. Finally, the compute fabric in particular transmits tensors, which are not semantically meaningful on their own. Together with a manifest (including the processed tokens), they would be, but the frontend fabric carries such information more directly.

This phase runs at line rate online, all the time. Nothing in it depends on an event request being singled out for analysis.

1. The request enters a monitored unit[8] over its frontend link and is served, tokens flow back out over the same link (typically wrapped in JSON files). The tap sees the traffic crossing the boundary: this request's bytes, its response tokens, and everything else (SSH sessions, file loads, orchestration, other tenants' traffic).
2. The tap hashes payloads over network packet-groups, unaware of payload object boundaries or contents. Timestamps are not sent out by the tap, as they may encode information in unverifiable trailing bits. Instead, timing is done by the prover and the verifier, when receiving the hashes.
3. The hashes leave over a fiber link that is split and sanitized (output cross-check 4.3.2, sanitized egress 4.3.3), so prover and verifier observe identical egress with timing and physical-layer side-channels scrubbed. Both parties store these hashes.
4. The prover recomputes the same hashes from their own copy of the captured network traffic, confirming the tap emitted nothing but legitimate commitments.
5. The prover retains the plaintext (or keeps it reproducible) and writes a manifest.

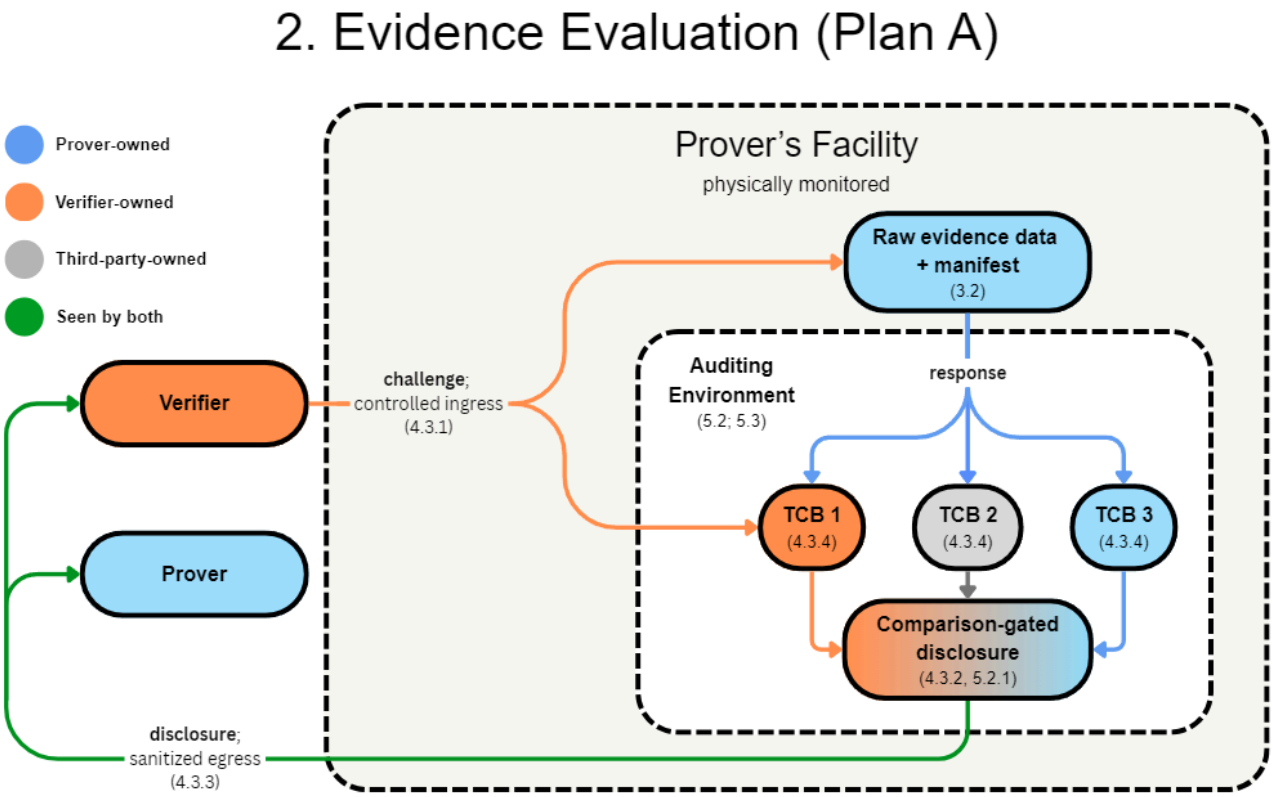
Optionally, memory challenges (section 5.1.2) are a second, independent evidence capture mechanism for ML compute use. Their commitment mechanism is analogous to that of the network taps.

Questions for the reader #5

A lot of the network traffic in a frontend link is not related to inference or training workloads, even when tapping right at the GPU server, at the lowest level of the network hierarchy. How feasible is it to require that all egress traffic is explainable by the ingress? Which engineering compromises need to be made? (Before bringing up non-determinism, read section 5.2.2).

[8] How large a unit is, meaning which backend network and compute domains are self-contained inside one air gap, is negotiated between prover and verifier.

3.2.2 Evidence evaluation



As long as the hashes and their preimages are in storage, the verifier can challenge random samples at any future point in time. In plan A, evaluation is performed in an isolated environment containing redundant computing platforms receiving identical inputs, and checking each other's outputs. Plan B removes the need for such a secure environment and replaces them with cryptographic techniques such as Zero-Knowledge-Proofs.

1. The verifier issues a challenge over controlled ingress (4.3.1), a one-way link enforced by a data diode and passive observability through which the prover can monitor the backward direction.
2. The prover supplies the preimages: plaintext slices matching the challenged hashes, the relevant manifest entries, and metadata required for re-computation (see section 5.2.2).
3. Each party's Trusted Computing Base (TCB) independently reassembles the challenged objects and checks them against the manifest and the challenged hashes.
4. Then each TCB verifies the inference workload by re-executing the supposed computation under the declared metadata.
5. The reconstructed workload is then screened for compliance (inference vs. training, model on the whitelist, outputs free of blacklisted use; inspector agents 5.2.3).
6. The TCBs cross-check their verdicts (4.3.2). Agreement is required before anything is released.
7. The result is released through sanitized egress (4.3.3), split to both parties. By default the disclosure is minimal, e.g. only the challenged hashes if there is no issue. In case of non-compliance, the disclosure could point to the hashes of the suspicious data.

Three conditions end the trace early, and they are distinct.

- A challenged preimage that is missing or does not hash to its commitment means the prover has failed the challenge outright.
- A re-computation that does not reproduce the committed output means either the declared metadata was false, or there was technical a problem in the setup, or the workload was not as claimed, both warranting investigation.
- TCB disagreement means no disclosure leaves and the disagreement itself triggers investigation.

What “investigation” means is a partly political question, which was raised in section 2b under “The attribution problem”. I treat it as out of scope here.

Faults, if triggered on purpose, can leak one bit of information to the verifier, per event. The prover and verifier need a pre-agreed budgeted rate for such faults, beyond which suspicion is raised.

4. Subsystem designs for eliminating the need for mutually trusted silicon

I recommend that the field strongly de-prioritizes trying to make general-purpose processors or AI accelerators that can be trusted by two rival nation states. Instead, I propose systems that use redundant checks across *unilaterally* trusted devices, and ways to build and red-team such systems. Mutual security is achieved via such redundant checks, and data movement control via inspectable analog components.

4.1 Trust in silicon is hard

One approach for verifying compute use would be to try to design a custom chip with transparent design, supply chains and tooling. Supply-chain hardened FPGAs exist and are used in high-stakes applications such as air-defense comm networks.

But OC5 adversaries are extremely sophisticated and resourceful. Proving a negative is inherently challenging here, especially considering the impressive creativity of attackers: In one example, researchers have demonstrated hardware trojans that work by altering dopant distributions during fabrication. No added gates, no changed layout, nearly impossible to detect even with electron microscopy.

Even if a processor such as a CPU, ASIC or GPU was secure, there are upstream challenges with securing the integrity of information movement: When a network tap intercepts light, its transceivers needs a sophisticated digital signal processor to turn light amplitudes into bits. Supply-chain hardening such processors for even unilateral trust by natsec agencies takes

years and expensive infrastructure. For three decades, nuclear arms control researchers have tried to build mutually trusted electronic systems for nuclear treaty verification, and failed at anything more complex than extremely minimal ASICs. As one review put it, "there is at present no accepted method for proving the absence of vulnerabilities in an electronic system."

4.2 Analog data movement control: passive splitters, data diodes, enclosures

Here I discuss the only parts of the architecture that need to be inspectable by both parties. They control which device sees what information, by duplicating data streams, ensuring one-way data flow through a wire, and preventing A) bypassing of network taps/gateways and B) exfiltration from confidentiality boundaries.

1. **Traffic duplication:** Optical fiber splitters clone traffic of one wire into multiple. They exist in a range of variants, but the simplest one (Fused Biconical Taper) works by fusing the glass cores of two (or more) fiber strands together. In plain English: Glass touching glass, which is inherently simple to inspect[9] against tampering.
2. **Data diodes:** One-way data movement can work in a variety of ways. In example, optical isolators prevent backreflection into lasers, but can also double as data diodes. Section 4.3.1 describes a diode setup that can be used even when there is suspicion of side-channels through the blocking direction.
3. **Enclosures:** There are established standards for SCIFs (Secure Information Facilities), but what is less established is "what defenses can be retrofitted at a massive scale to prevent bypassing of network taps". More on this in section *5.3.1 Side-channel defenses*.

Both the prover and the verifier can inspect the optical components, which should be tractable, as one only needs to confirm that the devices are analog-only appliances with no logic to collude with. If this requires deep and destructive inspection, the components should be hot-swappable, so that they are ready to be challenged and taken to a lab at a moment's notice.

4.3 Building blocks for a mutually secure verification system

Here I discuss how analog data movement control can be used to process information in a mutually verifiable manner, even between two covert adversaries who distrust each other's hardware. *Unilateral* trust in individual processors is still needed: The prover and verifier each have their own trusted computing base (TCB).

I describe these systems by function: Secure ingress and egress channels, cross-checks of device outputs, and offloading critical security-relevant tasks from performant-but-vulnerable hardware. The solutions I discuss here are *examples*, and the same tasks could be achieved in other ways. What matters is that such systems can be built and red-teamed quickly, which I think can be done for the systems discussed below, using off-the-shelf commodity hardware

[9] (with a microscope)

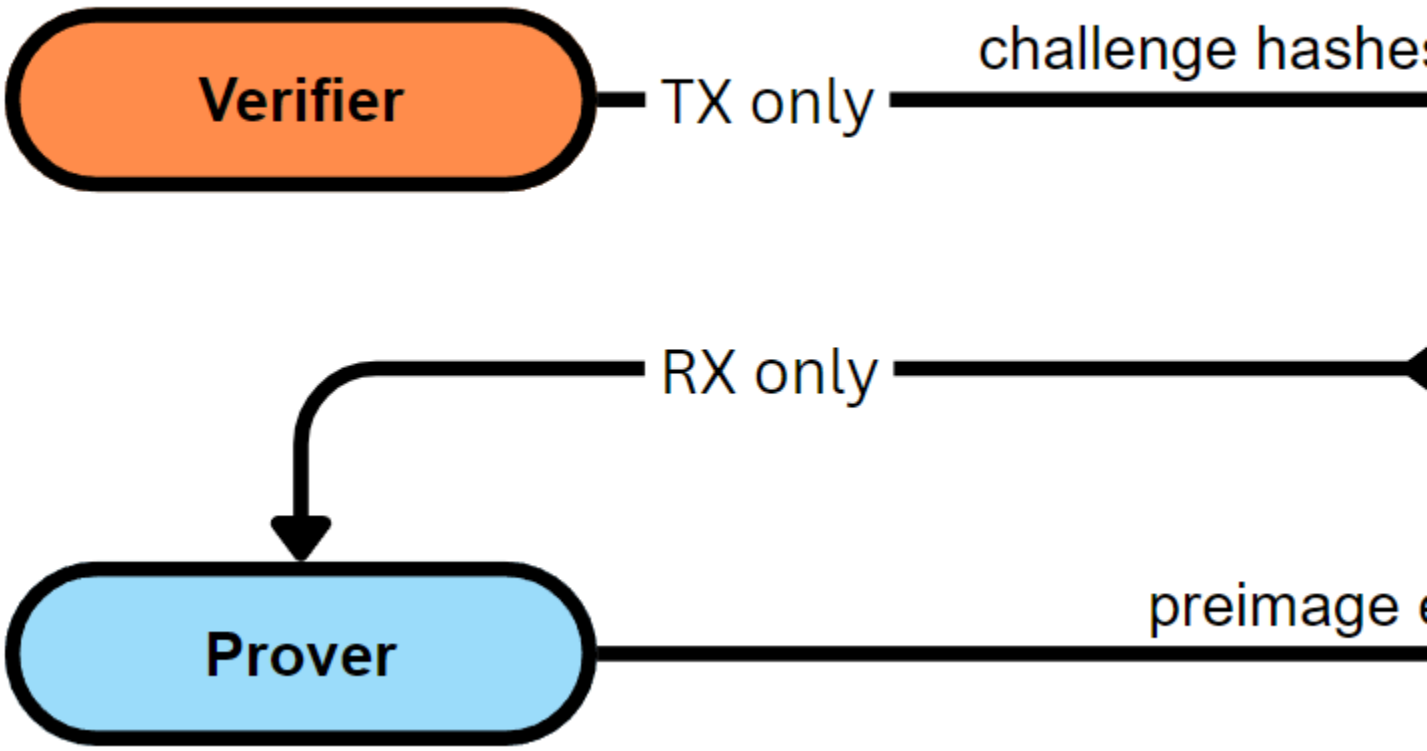
(with the possible exception of the instructor TCBs in section 4.3.4, which may need to be custom FPGAs).

4.3.1 Controlled ingress

Used in: the verifier's inward communication link for sending challenges into the auditing environment (the orange link in section 3.2.2's illustration)

This subsystem guarantees that a communication link is one-way only. This is especially important at the communication link through which the verifier enters challenges into the auditing environment.

While an optical isolator theoretically blocks light (of a given wavelength), one can raise the bar against side-channels by requiring that the receiving device does not include any dedicated emission hardware (e.g. the laser in a transceiver). On the other end, the emitting device should only be a laser, no RX. Finally, the prover can use passive splitters to observe the ingress into the data diode, and its egress. There should never be a signal passing though the blocking direction, and this way the prover can directly detect if and when this does happen regardless.

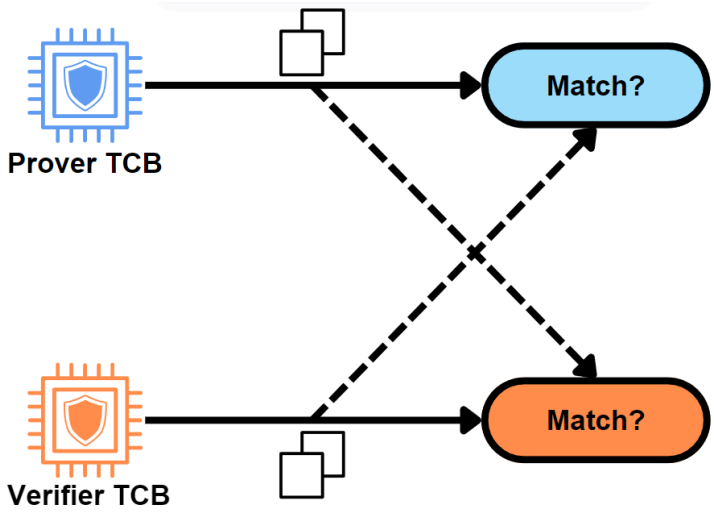


4.3.2 Output cross-checks

Used in: the network tap's egress checked by the prover, the auditing environment's TCBs cross-checking each other

In the reference architecture, there are at least two points where the prover's confidential data will be processed by devices that the verifier relies on (meaning they are verifier-trusted, prover-distrusted): The network taps and the auditing environment. Unless the prover can confirm exactly what enters and leaves these devices at all times, this is a dealbreaker.

To confirm that the verifier's device sends out nothing but the correctly computed result, the prover can clone the egress of that device with a fiber splitter, passively observing the exact same data stream that the verifier sees. They can compare this with their own re-computation. Example: The verifier's network taps send out hashes of observed network traffic, over optical fiber. This egress is split and fed back to the prover, who can re-compute these hashes and confirm that the verifier's taps behave correctly. The auditing environment applies the same principle: No disclosures leave the verifier's compliance screening device, without being passively observed and cross-checked by the prover.



Of course, analog and digital signals are not the same. With distrust in the verifier's optical transceivers, there remains suspicion of side-channels. The timing may leak information, or there may be undisclosed wavelength modulation etc. This problem is addressed by the next component.

Prior work

- N-Variant Systems implements redundancy in software: it runs diversified versions of a program and withholds any output until results agree. Compromising the computation would require a single input that simultaneously defeats every variant. Redundancy and diversity in the *computing substrate* or orchestration software is out of scope here.
- SAFER PATH explicitly addresses the threat model of secure computation despite suspected hardware trojans. The authors propose a small trusted computing base that mediates program execution across banks of untrusted COTS processors. The intended

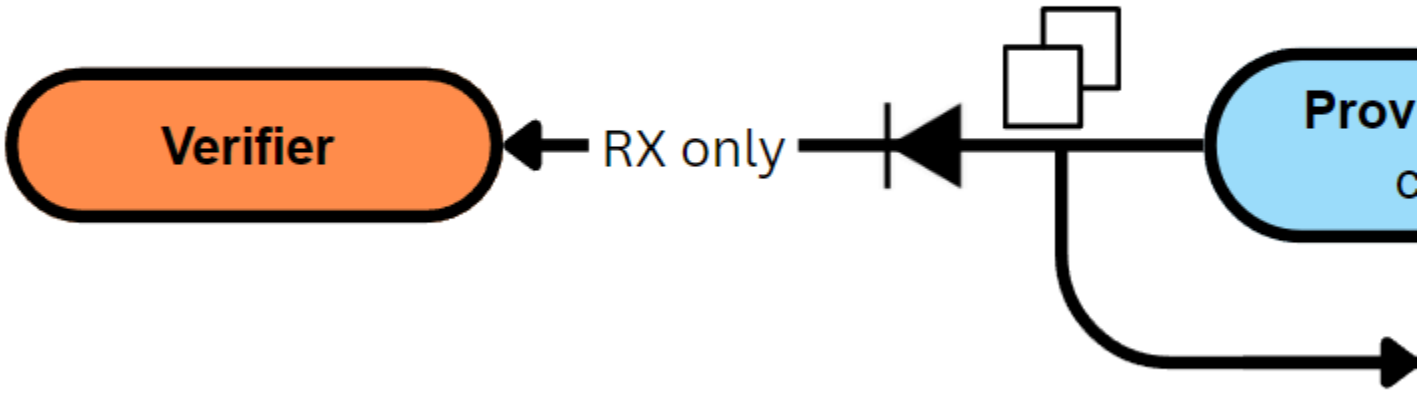
security model explicitly relies on physically diverse processing elements, though the prototype demonstrated the architecture with 30 identical cores on a single FPGA rather than diverse hardware.

- Verde compares outputs of language model inference across different hardware setups (CUDA-based on several NVIDIA GPUs), controls the order of floating-point operations and uses FP32 operations to achieve bitwise identical results. In section 5.2.2, I go into more detail regarding the determinism challenge.

4.3.3 Sanitized egress

Used in: any traffic sent to the verifier, including the network tap's egress and the auditing environment's egress

The prover needs to ensure that only the pre-agreed information reaches the verifier, and this in turn requires a way to sanitize away any physical or timing side-channels. One way to do so is to have a prover-controlled set of transceivers perform active forwarding: Conversion from optical to electrical and back to optical (OEO conversion). Any malicious optical modulation performed by the verifier's device does not pass through the non-optical medium. The re-emission can also happen at batched or randomized timing, defeating timing side-channels as well.



But since the verifier distrusts any traffic sanitization performed by the prover, the trust problem inverts. Again, optical fiber splitters are a way around this, as they can duplicate the sanitized egress back to the original sender of the un-sanitized signal. The verifier's device can compare what it sent with what was forwarded by the prover's OEO gateway. Data diodes can help enforce the correct direction of information transfer, and be monitored as in 4.3.1 if needed.

Prior work

- PHY covert channels describes the timing side-channel and covert bandwidth figures (if left unmitigated).
- Yen & Joye discuss faults as a potential leakage channel: A malicious device can leak one bit of information by deliberately outputting a wrong result, preventing a disclosure from leaving the system because the cross-comparison fails. Also, the timing of a disclosure can reveal information. This motivates a maximum tolerance for faults caused by the verifier's device, beyond which investigation would be triggered.
- **Active wardens** erase side channels in a communication link, including timing and network header steganography. This becomes important when disclosures need to be transmitted. See "prior work" under section 5.1.1.

4.3.4 Secure performance offshoring

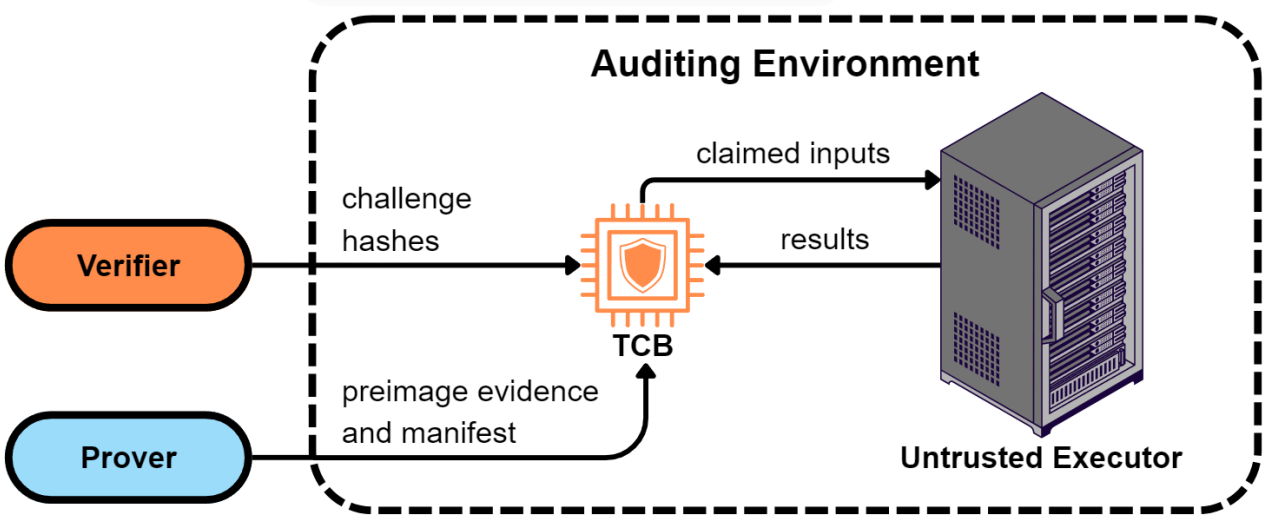
Used in: the auditing environment's replay devices. Each party runs their own instructor-executor setup.

While mutual trust in silicon is borderline impossible between rival nations, achieving even unilateral security against software-attacks of all kinds is not exactly easy. Formally verified logic is the only way of processing information that meets this standard, and only very minimalistic computing bases shrink down the attack surface sufficiently. If the task is as simple as comparing bits, managing memory slots or parsing file headers, one can feasibly construct the hardware from FPGAs or similar. Same goes for checking whether a piece of code has been signed by the prover and verifier before using it, even if executing the code is a complex task.

But the auditing environment DOES need to perform complex tasks at sufficiently high performance:

1. Re-assembling network packets and their payload objects from raw bitstreams; Checking if the prover's manifest matches these objects.
2. Screening this data using arbitrary programs that the auditing environment can receive anytime.
    a. Replaying ML workloads such as inference forward passes or training steps.
    b. Applying compliance screening software to generated ML outputs (see section 5.2.3)

The requirements of security (through simplicity and inspectability), performance and versatility are at odds with each other. However, one can separate performance-optimized hardware (the executor) from a dedicated, security-focused trusted computing base (TCB; the instructor).



Suppose a network tap or memory probe has captured an activation tensor, and this piece of information needs to be attributed to a compliant ML workload (e.g. inference on a given set of tokens). A verifier-controlled recomputation and comparison with the prover's claim are actually two separate tasks:

The security-critical task is the *comparison* between the claimed tensor and the verifier-controlled replay. If the verifier can not trust hardware that is performant and complex enough to re-execute frontier AI inference, they can manage an *untrusted replay device*. Only the claimed *inputs* of a computation are forwarded to this device, while the claimed *output* is deliberately omitted. That is to say, the untrusted executor is never informed about what claim the prover made. A compromised executor that knew the target could fabricate a matching tensor without running the computation, and blinding removes that shortcut so the only way to pass is to compute correctly.

The generated result is returned to the verifier's TCB, which compares against the claim. This disaggregation between an instructor and an executor makes it viable to use off-the-shelf AI accelerators securely, to re-run heavy workloads in practical timeframes, while still rooting trust in minimal, unilaterally secure TCBs rather than needing to secure complex ML accelerator technology stacks.

5. Engineering approaches for evidence capture and evaluation

5.1 Evidence generation, capture and commitment

The prover performs ML workloads on their hardware, and any results sent out via the frontend/in-band uplink is tapped. The prover can also create a record of claims, with supporting evidence such as a down-projected activation tensor at the last token position of a decode generation. Such evidence can then be appended to a manifest, a record of claimed hardware

activity, which needs to be consistent with the data captured by the verifier's taps and memory pingers. The prover, knowing exactly what hashes have been sent to the verifier, can create a public version and a private version. Both versions make claims about the hashes, which can be verified in the auditing environment (or via a ZKP, section 5.2.4).

- The private manifest can include fine-grained evidence such as tokens and the aforementioned activation tensor mappings. This is the claim that the AE (or the ZKP) would verify via re-computation and compare against the captured evidence.
- The public manifest, if the prover chooses to create one for the verifier, would only contain high-level labels. Example: "hashes #203197344530 to #20319734778 are a prompt completion, and hashes #203197344778 to #20319739011 belong to a file download."

~~In principle, network taps and memory probes are complementary ways to interrogate hardware activity and data: One captures flow, the other probes state. The focus on network taps as the default is rooted in A) the fact that network taps can be used to probe the memory of devices below the tapped level B) probing small memory domains requires closer proximity to monitored memory, and therefore more complex and expensive installation.~~

In principle, network taps and memory probes are complementary ways to interrogate hardware activity and data: One captures flow, the other probes state. The focus on network taps as the default is rooted in the fact that probing memory domains requires (much) closer proximity to the monitored memory compared to traffic monitoring, and is therefore more complex and expensive to install.

5.1.1 Network taps and active wardens

Network taps, tap aggregators, packet brokers, and packet-capture systems duplicate network traffic for timestamping, analysis, compliance monitoring, or storage, usually without disrupting the production link. They are already used in non-AI compute governance and security contexts, including stock exchange-network monitoring[10], enterprise/security packet capture[11], defense-network visibility[12], and lawful-intercept[13] in telecommunication.

Transferring this to the use case of

[10] Deutsche Börse's Frankfurt deployment used 60 Arista 7130 devices to tap, aggregate, timestamp, buffer, and capture exchange-network traffic. The regulatory context is MiFID II, which requires timestamped evidence capture for legal auditing.

[11] At the time of Endace's December 2024 DoDIN APL certification, OMB Memorandum M-21-31 required U.S. federal civilian agencies to improve cyber logging and to retain full packet-capture data for 72 hours under the highest of its four event-logging maturity tiers. In May 2026, OMB rescinded M-21-31 and replaced it with M-26-14, a risk-based, prioritized logging and network-visibility mandate.

[12] Network monitoring at the physical, rather than the application layer,

[13] For CALEA compliance in the U.S., Subsentio's Safe Harbor Probe, a passive appliance connecting via network taps or SPAN ports, is deployed at roughly 100 US service providers. For ETSI-compliant lawful intercept in the EU, Keysight's deployment uses both physical and virtual taps.

requires some adjustments. For this, we first point to related work:

Prior work

- The Fundamentals and feasibility of secure network taps for verifying AI datacenter use This post presents the option space for where and how to tap into AI datacenter networks, what technologies are available, the feasibility of different approaches and open technical challenges.
- Network Tapping for AI Verification: A Technical Assessment Amodo design has picked up research on network taps, as exemplified in this post. Here, different tap technologies are compared, with a focus on integration concerns around link budgets, reliability and latency. Amodo is continuing this research and expanding their engineering team.
- NetWarden: Mitigating Network Covert Channels while Preserving Performance Researchers at Rice University built an "active warden" based on a programmable switch, which erases potential covert communication channels in network packet headers and timing. The authors claim it is the first practical high-speed device of its kind.
- Hardware acceleration design of the SHA-3 for high throughput and low area on FPGA FPGAs are a common platform for network taps. In our case, they need to hash network traffic at high line rate, and do so with minimal logic, ideally without software. This is a mostly solved problem research-wise. I am not aware of any commercial products, though.
- Fingerprinting All AI Cluster I/O Without Mutually Trusted Processors This paper presents an architecture for a "trustless network tap", which transmits hashes that are double-checked by mutually distrusting processors. It also integrates a trustless active warden, with a similar redundancy principle.

The basic idea is to group untrusted devices into a perimeter where only monitored links communicate in and out.[14] Network taps transmit only cryptographic commitments of the captured traffic to the verifier, while the plaintext is never shown to them. For inference clusters, the perimeter can be as small as individual pods of a few hundred accelerators each. This would keep model parallelism and KV transfer self-contained within the unit, not requiring such high bandwidth communication to traverse any tap. The tap would capture the orchestration traffic, file loads in and out of the pod, and, most importantly, the workload-identifying traffic such as tokens. Monitoring training requires either larger perimeters (potentially as large as the whole cluster used for a training run), or network taps that can capture all-reduce on the compute fabric and checkpoint transfer on the storage fabric, both of which are very high-bandwidth and likely warrant optical random sampling to be viable.

[14] Optionally, traffic between devices inside the perimeter (East-West communication) can be monitored as well, but storage and compute fabrics (NVIDIA terminology) typically carry tensors rather than tokens. For more details, see the discussion of East-West traffic monitoring in "The Fundamentals and feasibility of secure network taps for verifying AI datacenter use"

Open research questions

- The actual composition of traffic entering and exiting an AI datacenter. The cost estimate of North-South network taps calculated in my post rests on the information bandwidth being not more than two orders of magnitude larger than the expected API traffic at full inference utilization of the monitored hardware.
- While there is commercial technology and precedent to build on, the redundancy architecture (as for example in the SeGaDev) needs engineering validation and red-teaming. For example, in case of a disagreement between mutually distrusting processors, what alarm channel can be sufficiently secure against a nation-state covert adversary?
- The (to my knowledge) most advanced active warden built so far, by Xing et al., was a re-programmed Tofino switch. A treaty-setting would likely require something more circuit-level configurable and resistant against supply-chain attacks, such as a purpose-made FPGA. An active warden hardwired into FPGA logic has not been made yet.
- What would be an ideal batching heuristic for hashing and signing, i.e. how many network packets/Bytes per hash? The prover and the verifier would need to agree on a fixed scheme.
- Empirical feasibility of passive optical splitting at 53–112 GBaud under realistic conditions: This is a question Amodo Design is actively pursuing. This issue determines whether storage-fabric and compute-fabric taps are deployable quickly, at realistic cost and at a large scale, which in turn determines whether or not North-South taps should be the sole focus, or if East-West photonic random sampling can work as well, using passive taps.
- How to securely install the network taps, and how to supply them with power?

5.1.2 Memory challenging and memory wiping

A memory challenge is a timed question a machine is made to answer about its own information content. In our verification context, this can be useful in two ways: A) verifying presence of information B) verifying the absence of free memory/storage. The latter is an implementation of the former, where the information capacity of the device is known, and the challenged information fully occupies the space.

A) Verifying presence of information

The general idea is for the verifier to challenge a part of the data the prover claims to hold in memory. Response time is the primary evidence against evasion[15]: Responding from another device can be measurably slower.[16] Because ML hardware is tightly interconnected, it is more

[15] Evasion options: outsource storage, recompute on demand, outsource computation, or disable the checker.

[16] Of course, timing would need to be measured by a verifier-controlled device. Since network taps with precise timestamping functionality are already part of the reference architecture, modifications could be made such that they could either double as memory challenging devices, or sit between the pinging device and the challenged servers, hashing the challenges and responses in the same manner in which all network traffic is captured regardless of the content.

useful to think of "**response-time domains":** a domain is any set of locations whose mutual latency differences fall below the measurement resolution *at the location of the pinging device*. Details depend on the deployment (e.g. whether DRAM from the closest server can be distinguished vs. RDMA from another[17]). DRAM responds faster than any disk (~100ns), including high-end NVMe SSD.[18]

B) Verifying absence of free memory/storage

With knowledge of the memory and storage capacity of a device, one can use memory challenges to verifiably wipe data. Incompressible noise can be loaded into the device, until all DRAM (and disk) capacity is full. Random samples can then be challenged as described above. This introduces some down time, which is mostly dominated by the writing speed and storage of on-board disks. For NVMe SSDs, this should be tens of minutes. Diskless servers could of course be wiped much faster. Verification of memory/storage saturation depends on the ability to rule out RDMA, either via response latency or physical disconnection.

I consider memory challenging as an *optional* evidence collection mechanism in addition to network taps. Depending on the resolution at which one wants to distinguish memory domains, it can either be a slight modification of the network taps already in use, or an expensive system of one probe per server.

Prior work

- Computational accounting (in submission at TAIGR): fills all DRAM/HBM/caches with a verifier bytestream and challenges random indices under a deadline (**this is case B / wiping**). Probe: an external auditor over the network, basically a **network tap**.
- Memory-printing: Domain boundary is *RAM vs flash*. Fills free RAM with a keyed PRNG in scattered order, so any resident attacker is forced into the slow flash domain and caught by the delay. Probe: external verifier (base station), or preferably an on-device SIM card, since network timing variance is too large. LightSec: Same *RAM-vs-flash* domain. Probe: **on-device SIM card**; its serial link is slow but low-jitter, which is the property that matters.
- SAGE: Domain boundary is *on-GPU vs off* — a checksum kernel saturates all SMs and registers so the data must sit in HBM. Probe: an SGX enclave on the same host, timing the GPU **one PCIe hop away**.

---

[17] Assuming the probe sits right at the in-band uplink of the challenged server, the timing noise of the NIC between the host and the probe decides whether local vs. remote DRAM can be resolved. That jitter is tens-to-low-hundreds of ns *if* the challenge-response path uses kernel bypass / polling rather than stock kernels (PCIe-bench: 99.9% within 80 ns). The challenge is unpredictable, so the suspected server must request the data from the holder and wait for it to come back. A full RDMA round trip in the critical path, ~1–2 µs over InfiniBand/RoCE (NVIDIA ConnectX-6: sub-800 ns; RoCE HCA as low as 1.3 µs) versus a local DRAM read of ~70–200 ns (Shuhai, FCCM 2020). So the ~1–2 µs signal clears the noise by ~10× on a tuned path. The remaining evasion is pre-staging the data into local memory before the challenge, which an unpredictable, capacity-filling challenge (case B) closes.

[18] Intel's P5800X brief calls it "the world's fastest data center SSD," says average read response times remain below 6 µs, and gives 3 µs average latency for 512B random reads. Compare this to DRAM reaction speeds of around ~100ns.

- TrustCheckpoints / MULTIPASS: Domain boundary is *on-chip SRAM vs off-chip* — a k-independent randomized polynomial whose space–time bound forces any extra payload off-chip. Probe: a trusted external microcontroller on a dedicated bounded-jitter link (explicitly needs sub-µs/ns jitter), i.e. **physically adjacent to the device**.

Note how most of these mechanisms require a probe in **close proximity** to the challenged memory.

Open research questions

- To my knowledge, distinguishing content of DRAM (HBM, DDR) between one server node and another, using a challenge+response timing probe on any network level, has not yet been demonstrated. The probe would need a reporting channel to the verifier, a specific and somewhat generalizable protocol, and nonce-generation the prover can not predict. I expect that one such probe per NVLink scale-up (or comparable systems) would cost far less than the monitored hardware, but like all other components it needs to be adversarially robust.
- A generalizable software-protocol for challenge-response, covering enough diverse data types to allow mostly normal operation for the prover.
- Measurements of the reaction times of different memory locations, measured from various locations in the network: Right at the server's transceiver, behind a few switch-hops, at a PCIe port inside the server…
- Threat models and red-teaming: What are the covert evasion measures that are not suspicious even for repeated memory challenges? What evasion is possible with non-eraseable residual capacity in NAND etc?

5.2 Evidence evaluation and disclosure

5.2.1 Secure auditing environments (tentative plan A)

On a high level, the auditing environment (AE) receives the verifier's challenges, and the prover's plaintext data. In the standard configuration of the reference architecture, this is the captured network traffic data and its cryptographic commitments signed by the network taps. Since the architecture concentrates all computational complexity here, rather than at the evidence collection stage, the cyber-attack surface is larger by default.

Unilaterally "trusted" computing does exist, and remember that the threat model is a *covert* adversary against whom redundant detection methods can be layered. Finally, the AE only needs to compute on a small subset of the prover's records, and can take advantage of not having to serve customers at comfortable speed. All this points to air-gapping, tight I/O control, and redundancy between mutually distrusting computing hardware/TCBs. In section 4.3, I described the building blocks from which such an AE can be constructed: Controlled ingress, multiple instructor-executor setups with output cross-checks across the different party's TCBs, and sanitized egress.

The recommended implementation works as follows:

1. At least three distinct TCBs (different supply chains, hardware and software. One trusted by the prover, the verifier and a third-party) make up three small clusters, all inside the prover's datacenter and inside their own air-gap.
2. Any inputs sent by the prover and the verifier are split and fed into each device.
3. All outputs off all devices are split and compared before they leave the air-gap. When leaving the environment, the outputs are split again and shown to both the prover and the verifier.
4. Like the prover's hardware, the evaluation hardware (including all I/O links) is physically monitored. For details on air-gaps against *covert adversaries*, see section 3.3.1.[19]

The security of redundant cross-checking hinges on the different devices *not* having the same vulnerabilities, which means they need to be built from independently sourced components, and not share (too much) code or architectural similarity in general. Even with different hardware and software, outputs can (and must) still be identical. This is compatible with a requirement that the outputs would most likely already have: Being very minimalistic, as to not offer degrees of freedom for covertly encoded information. And in case of a dispute, the AE's outputs could be limited to only the hashes of the non-compliant data.

Prior work

- **Evaluation environments under bilateral distrust:** Arms-control verification has precedent for dedicated AEs that process sensitive evidence while restricting inspector-visible outputs.
    - CIVET is a fully inspectable hardware/software information-barrier system for analysis of sensor data read from nuclear warheads: it analyzes detector data with *jointly* developed software on simple verifiable hardware and releases only non-sensitive go/no-go conclusions to inspectors.
    - TRIS is a similar system, but uses custom hardware for one specific logical operation, analyzing a compressed radiation spectrum in an isolated environment.
- **Comparison-gated output:**
    - The most relevant existence proof I found is TrustGuard / CAVO, and its follow-up piece. The former introduces the CAVO model (**C**ontainment **A**rchitecture with **V**erified **O**utput) and the **sentry** sitting between an untrusted computing system and external interfaces. The latter adds updated evaluation and prototyping results. The sentry verifies and gates egress by checking the

[19] Note that there is one key difference in requirements: At the prover's servers, an upper-bound on covert bandwidth can make undisclosed use of ML accelerated work infeasible. But the evaluation environment contains *programmable* devices controlled by the verifier, which themselves work on model weights, customer data and training data. This means that even single-digit bits of leakage can be consequential. While the side-channel defense is stricter than at the prover's servers, the evaluation environment itself is small (possibly as small as three server racks), which means it is more practical to guard it with extreme caution, than it would be for a whole datacenter.

processor's committed execution trace: for non-memory instructions[20], it re-executes the relevant arithmetic and control executions on its own functional units and compares them against the reported results. This assumes *one* trusted sentry, but the reference architecture gates output with *redundant* cross-comparison performed by *multiple* parties. Still, Trustguard sets a precedent for egress control via re-execution on a distinct, minimal TCB: the authors validated it in simulation and prototyped an FPGA-based sentry with only 3,543 HDL[21] lines worth of logic.

- In aviation, railway management and spaceflight, redundant computation and comparison-gated output is an established practice, though the typical use case is fault tolerance rather than redundancy and diversity against cyberattacks.

Open research questions

- Physical security in every monitored datacenter is challenging when defending against covert communication between the prover and their hardware, but much more so when defending against the verifier: The higher the number of evaluation facilities, the more failure points exist that could leak information. In contrast to the task of preventing leakage to the prover, there is practically no tolerance for leakage to the verifier.
  - From the perspective of the security of the prover's secrets, it preferable to have AEs
    - A) within each of their facilities (such that not even encrypted sensitive data needs to leave),
    - B) fewer environments, which would warrant a secure data transfer protocol out of the prover's datacenters, to a shared AE.
  - In a treaty scenario, what protocol would the responsible authorities deem secure enough?
- What leakage budget, if any, is acceptable from technical failures and retries?
- How to securely install the unilateral devices into the multilateral evaluation and cross-checking system?
- How to supply power to the AE securely?
- How to interact with the eval environment securely? I.e. how to feed in data, code and instructions? Some device has to produce the input, the TX feeding into the optical fiber before the signal splits into the RX of each party's device in the AE.

---

[20] Memory operations are checked differently: TrustGuard avoids duplicating the whole memory subsystem by using MACs and a Bonsai Merkle Tree to verify cache-line integrity and freshness, while output operations are held until all relevant instruction and memory dependencies have been verified. The reference architecture does not rely on authenticated memory outsourced to the same untrusted processor being checked, and does not rely on using a verification protocol that depends on guarding a key from the untrusted processor. Instead, all devices use their own, independent memory and processing. However, the sentry buys a critical property by re-executing *only* the arithmetic and merely *verifying* memory integrity: while the untrusted processor performs all the instruction preparation, scheduling, and memory management that dominate a processor's complexity, the sentry's 3543 LoC are amenable to formal verification.

[21] Hardware Description Language

5.2.2 Replay and the determinism challenge

When it comes to evidence of ML workloads, there are two categories of verification: one validates that a claimed computation is accurate, in the sense that the claimed computation indeed produced the claimed output. The other checks whether this computation was compliant with rules. Here, we focus on the former challenge, while the next section goes into the latter. If re-computation can not produce the exact same output as the claim, problems emerge: Firstly, the traffic observed by a network tap is not fully verifiable, meaning that there are degrees of freedom in network traffic that can carry information covertly. Secondly, redundant replay in an evaluation environment may produce non-identical output across different processors, which would create disputes.

The reference architecture needs precise replay, without enforcing (too) expensive limitations on the prover's training and inference. This is achieved by having the prover avoid arithmetic under race conditions, and having them track the properties affecting numerics:

Specifically, this includes:

1. **Hardware SKU** (determines tensor core arithmetic and special function unit behavior)
2. **Exact model weights** (in the deployed quantization format)
3. **Parallelism topology** (separately for prefill and decode stages)
4. **Software versions** and any custom kernels (CUDA toolkit, PyTorch, attention backend, quantization library)
5. **Batch size** at each forward pass

Among these properties, only batch size changes dynamically during inference. Either logging the batch size (a single integer) or recording whole batches in their original composition resolves the problem.

With this information, the evaluation environment can re-compute the exact arithmetic of the prover's original computation. Identical hardware is not required, only information needed to replicate reduction order. However, an instructor-executor setup as described in section 4.3.4 is still an option for re-executing on identical hardware, even if such hardware is untrusted.

Prior work

- Bit-Exact AI Inference Verification Without Performance Tradeoffs: My case for the claims above.
- Verifying LLM Inference to Detect Model Weight Exfiltration: Statistical analysis of non-identical replay, upper-bounding possible covert bandwidth by orders of magnitude compared to the previous state of the art.
- DiFR: Inference Verification Despite Nondeterminism: Efficient replay in prefill mode, verifying quantization, or any other property of the model that is at least as numerically sensitive to change.

Open research questions

- The emulator behind my paper is a proof-of-concept that was not optimized for efficiency. If replay runs on CPU, one can optimize the software and achieve high utilization. Emulation can be parallelized just as the original computation on the accelerator, across multiple processors. This is an open engineering problem, rather than new science. The more interesting question is what slowdown in wall-clock time due to emulation overhead is tolerable, which –above all– depends on the sampling rate.
- Since simulation can fully run in software, it should in principle be possible inside zero knowledge proofs (see section 5.2.4 for more details). How much does emulation of floating-point reduction multiply the (already high) relative overhead of ZKPs?

5.2.3 Inspection software, inspector agents

The most interesting verification objectives are those that not only check computational/quantitative properties (inference or training, number of FLOPs) but semantic properties: what inference is used for or not, whether a model architecture uses human-readable chain of thought, what kind of data was (not) trained on.

Checking whether a deployed AI system is on an established whitelist is an unambiguous and deterministic task. But deciding whether or not to whitelist a new model, or deciding whether model outputs are evidence of blacklisted use, is where judgement calls need to be made. The reference architecture does not prescribe which software and which evaluation criteria to use inside the auditing environment, but it can run arbitrary programs (see section 3, "*Programmable, retroactive compliance screening*").

Prior work

Automating semantic screening is not a new science strictly speaking, as internet platforms have been using content moderation software for decades. However, for deeper semantic understanding of the meaning and context of a computation, simple pattern recognition and keyword matching can fail to generalize, especially under adversarial circumstances.

- Inspector agents: Hausenloy and Li built an LLM-based inspector, a scaffolded Claude that reads a captured trace and returns a verdict, confidence, and cited signals. Their demo classified train-versus-inference and model size based on telemetry data (power, utilization, memory, clocks, temperature).
- Clio: Anthropic's Clio does semantic analysis of their customer's Claude usage, and only high-level summaries reach human developers. It catches coordinated misuse, but its outputs are deliberately not wired to automated enforcement.
- Agentic Witnessing is an inspector-agent protocol with an attached disclosure bound. An LLM auditor reads private files and returns only one of {True, False, Unsure, Error}. The software layer transfers to my proposed system, but the paper's authors rely on TEEs.

Open research questions

- **Adversarial robustness of inspector agents and classifiers.** Inspector agents need to be sufficiently resistant against prompt injection attacks, which are explicitly named in Agentic Witnessing. The false positive rate is an issue as well, which can create plausible deniability in incidents.
- **Classification precision.** A key question is whether the robustness of classifiers benefits from the same progress in AI technology which makes the monitored AI systems more capable.
- **Backdoor detection.** Adversarially trained AI systems may hide malicious action in steganography. Assuming training verification against backdoors has not reached maturity yet, can probabilistic deterrence against backdoor attempts work? If so, what probing methods and sampling techniques scale with increasingly more probing, data collection and evaluation?

5.2.4 Zero Knowledge Proofs (tentative plan B)

If cryptographic techniques such as Zero Knowledge Proofs (ZKPs) mature, they could remove the need for secure computing hardware setups, such as those discussed in section 4.3 and section 5.2.1. No verifier-controlled device ever needs to operate on the prover's plaintext data.

A ZKP lets a prover convince a verifier that a statement of the form "I know a witness *w* such that *C(x, w) = y*" is true, without revealing anything about *w* beyond what *x* and *y* already imply. *C* is a public computation (here, the forward pass of a specific model architecture at specific weights), *x* is the public input, *y* the public output, and *w* the private inputs and intermediate values. Since the reference system relies on random sampling rather than full re-computation, the right question to answer to assess the viability of ZKPs is "what statistical guarantees can you get with current ZKP techniques at what overhead?". Because of the favourable scaling behaviour of random sampling (see appendix A1), ZKP can already be viable if the goal is to upper-bound the possible percentage of false reports with a given confidence. **Verification becomes statistically useful once you can afford hundreds to thousands of checks per audit window.**

Prior work

Zero-knowledge proofs for ML inference have matured from toy demonstrations to systems that handle realistic models, though their computation remains expensive. Progress has been made in reducing the overhead, and further efficiency improvements can make ZKPs more viable for sub-sampled verification.

- ZKML was the first optimizing compiler to produce ZK-SNARKs for ML models beyond CNNs. It is computed on CPUs, but can be parallelized across multiple. The overhead is heavy: a 81M-parameter model with 189M FLOPs takes roughly an hour of proving on 128 cores and a terabyte of host memory.

- zkLLM is purpose-built for LLMs and computed on GPUs. The authors demonstrated CUDA-accelerated proving on a single A100 GPU for 13B-parameter Llama models. One forward pass on 2048 tokens was proven on this A100 in 803 seconds. Memory overhead has a factor of ~1, proving overhead scales sub-linearly with model size (exponent ≈ 0.5).
- Sheybani, Ahmed, Kinsy & Koushanfar survey 25 open-source ZKP frameworks across the four major constructions (zk-SNARKs, zk-STARKs, MPCitH, VOLE-ZK), benchmarking each on SHA-256 and matrix multiplication. The survey identifies hardware acceleration of proving as the most promising near-term path to closing the overhead gap, pointing to GPU-based multi-scalar multiplication (cuZK, GZKP) and FPGA/ASIC proposals as reducing prover wall-clock time by 1 to 2 orders of magnitude. It also notes that zk-STARKs and VOLE-ZK offer post-quantum security without trusted setup, at the cost of larger proofs.

Standard zkML systems (ZKML, zkLLM) hide model weights from the verifier but require the model architecture (layer count, kernel sizes, attention heads) to be public, because the architecture is encoded into the circuit structure. Guo, Qu, Guo & Zhang address this with *architecture-private* zkSNARKs for CNNs, where the architecture itself is committed rather than published. The only information leaked is the model's approximate scale.

Open research questions

- ZKPs run on integers. They do not (directly) compute floating point operations, but can emulate them. However, AI accelerated inference and training accumulate floating point rounding errors due to non-associativity, meaning that the outputs will not be exactly identical if computed without emulating the exact reduction trees.[22] Statistical checks such as DiFR can already compute pass/fail at high statistical confidence: “Was this sequence generated as claimed”? For bitwise precise replay, one would need to run a simulation of the original computation’s reduction tree, as I have demonstrated for Qwen3 4B in Pytorch/HF transformers. However, running such emulation inside a ZKP is an open, intricate problem.
- The more the efficiency of ZKPs improve, the higher the sampling rates become that are viable at a given compute budget, and therefore the statistical guarantees.
- Ideally, a ZKP would be not only efficient, but also architecture-private. Combining this with reduction tree emulation for precise reproduction of floating-point inference on accelerators would eliminate the need for trusted auditing environments as discussed in section 5.2.1.

[22] The perplexity gap is small and shrinks with scale: OPT-125M moves 26.56 to 26.65, LLaMA-2-7B 7.036 to 7.049, LLaMA-2-13B 6.520 to 6.528. The paper's summary is that the increase stays under 0.1 everywhere and falls below 0.01 by 13B.

5.3 Support mechanisms

5.3.1 Side-channel defense

While network taps need not be the only layer of defense (see section 5.1.2), it should not be trivial to evade them. A malicious prover may attempt to smuggle information in and out of their ML servers, past the network tap, via physical side-channels. The reference architecture critically relies on intercepting communication rather than directly securing and verifying computation on-chip, so this threat needs to be addressed.

Prior work

Side-channel research is a science of its own, and has been long before the recent surge of interest in AI. In contrast to many of the open research questions listed so far, this one is far less neglected, at least when it comes to the discovery of exploits. But when it comes to defense, and the specifics of our threat model (covert adversary attempting to run undeclared ML workloads on monitored hardware and exfiltrate results), related work has overlapping, but different goals.

Side-channel defenses need to be just as retrofittable, cost efficient, and scalable as the rest of the reference architecture. This problem motivated my recent survey on side-channels, and possible mitigations. While details are to be found in that report, I want to highlight the key takeaways which inform the reference architecture:

1. Suppressing (physically possible) covert bandwidth below kilobits per second is much more achievable than aiming for absolute zero.
2. For any physical information channel of interest (RF, acoustic, conducted, …), the key metric to target is the signal-to-noise ratio (SNR) of an attacker that could plausibly evade the verifier's sensors. The verifier can suppress SNR by either attenuating signals (enclosures, filters), or adding their own noise (jammers).

The highest-bandwidth side-channels to look out for are electromagnetic in airborne RF and power lines. I recommend prioritizing these channels via look-through jamming (as explained in my post), and low-leakage power line filters.

For such security research focused on AI in particular, I can point to Tampersec, the SL5 task force and of course the security teams at AI companies. Note that in this field, published work is likely to be just the tip of the iceberg.

Open research questions

- **How to construct a mass-manufacturable, "good enough" defense, particularly for power line filtering.** While there already are SCIF/ICD 705 , TEMPEST and comparable standards for how to secure information perimeters, most—if not all—currently available solutions address the threat model of an *openly* malicious adversary, against a defender guarding secret information contained in a perimeter. Furthermore, the practical requirements of existing solutions are different: They are much closer to the

zero-leakage goal, and much less focused on rapid scalability. For example, TEMPEST-grade power filters are low-volume products assembled and tested mostly manually.

    - **The viability of look-through jamming against different types of side-channels, and hardware-demonstration.** This research question is a "low-hanging fruit" for electrical engineers: Antennae, piezoelectric sensors, ADC, PCBs, basically household electronics. Counterintuitively, contemporary electronics are preferred over highly custom and expensive technologies, since this is about validation for mass-manufacturable solutions.
    - **Tamper-evident, rapidly mass-manufacturable and retrofittable enclosures.** Look-through jamming is active defense, enclosures add a passive layer. TEMPEST-grade SCIFs are inadequate for mass deployment.
- **Red-teaming and threat modeling.** What covert ML hardware use is practically possible at what covert side-channel bandwidth? Can a red-team successfully run undeclared workloads on the monitored hardware, at a significant scale, and exfiltrate results? This is an open-ended question at every stage upwards of the first prototype of the reference architecture.

5.3.2 Resource accounting

The evidence collected by network taps and memory probes can inform how occupied the resources of monitored devices are. Replay can even quantify the utilization of tensor cores, especially if it uses numerical forensics to precisely verify the reduction trees, and therefore batch size, kernel libraries and parallelism topology.

A simulation of the prover's hardware resources and their occupation (memory volume and bandwidth, compute utilization) can be cross-compared with the actual computational throughput (generated tokens in inference, or gradients in training).

Prior work

There is ongoing work on this topic from others, which I can add to the next iteration of this working draft, as soon as this work is published.

Appendices

A1. The statistics of random sampling

The amount of compute needed for e.g. 90% confidence of 0.1% maximum false reports does not scale with the amount of monitored compute. If a randomly sampled batch of devices (sample size n) is checked and every check finds a flaw if there is one, the random sample can upper-bound the total fraction of flawed devices (p) with a certain probability P(detect ≥ 1)

$$P(detect \geq 1) = 1 - (1 - p)^n$$

The finer the granularity of evidence reporting, the more favourable the statistics of random sampling. For example, if individual forward passes are re-computed, rather than whole sequence generations or batches, the samples are smaller, but not necessarily less informative for AI governance-relevant criteria.

Checked samples (n)	**95% confidence that p ≤**	**Detects 1% flaw rate with**	**Detects 0.1% flaw rate with**
100	2.95%	63.4%	9.52%
300	0.99%	95.1%	25.93%
1,000	0.30%	99.996%	63.23%
3,000	0.10%	~100%	95.03%
10,000	0.030%	~100%	99.995%

Verification becomes statistically useful once you can afford hundreds to thousands of checks per audit window.

Let's anchor on zkLLM's measured LLaMA-2-13B figure (≈803 GPU-seconds per proof, so ≈108 proofs per GPU per day), with no frontier extrapolation. Our reference is a 200,000-GPU fleet at $\approx 1.7\times10^{10}$ requests/day, where one proof re-runs one request as a single forward pass over its full token sequence.

What a clean daily audit certifies, by proving budget (current zkLLM, 13B-class proofs)

Prover GPUs	Share of fleet	Proofs/day (n)	Requests sampled	Clean audit rules out a non-compliant share above (95%)
1	0.0005%	~110	1 in 1.6×10^{8}	2.7%
10	0.005%	~1,100	1 in 1.6×10^{7}	0.28%
100	0.05%	~11,000	1 in 1.6×10^{6}	0.028%
1,000	0.5%	~108,000	1 in 1.6×10^{5}	0.0028%

Reading: at 100 dedicated prover GPUs, five hundredths of one percent of the fleet, a day with zero failed proofs certifies at 95% confidence that less than 0.028% of inference was non-compliant, while checking only one request in 1.6 million.

Scaling the model by two orders of magnitude:

Same statistics, ten times the cost. The certified ceiling is a function of the proof count n alone, so the bound columns are identical to the 13B table; what moves is the price of n proofs. Under the exponent zkLLM actually demonstrated (α ≈ 0.5), a 100x larger model means each proof costs 100^0.5 = 10x more (≈8,030 GPU-s instead of 803), so every overhead figure multiplies by ten. Holding the fleet at 200,000 GPUs but serving a model 100x larger in active inference scale also means the fleet emits ~100x fewer requests, so $\approx 1.7\times10^{8}$ per day.

Same audit, model scaled ×100 (α ≈ 0.5, ≈10.8 proofs/GPU/day)

Prover GPUs	Share of fleet	Proofs/day (n)	Requests sampled	Rules out non-compliance above (95%)
10	0.005%	~110	1 in 1.6×10^{6}	2.7%
100	0.05%	~1,080	1 in 1.6×10^{5}	0.28%
1,000	0.5%	~10,800	1 in 1.6×10^{4}	0.028%
10,000	5%	~108,000	1 in 1,600	0.0028%