Verifying AI Compute by Bounding Unexplained Information Exfiltration

James Petrie [1 2] **Yannick Mühlhäuser** [1 2] **Peter Drotos** [1]

Abstract

We propose an architecture for verifying AI compute agreements by bounding the amount of unexplained information in network outputs, without requiring either party to trust the other's hardware. Prover compute nodes are isolated so that all traffic passes through a verifier interlock, which records cryptographic commitments of the encrypted traffic. The verifier periodically challenges the prover to demonstrate that a randomly selected output is well-predicted by a policy-compliant computation on the measured inputs. The better a compliant computation can predict each output token, the less bandwidth remains for exfiltrating results from undeclared workloads. We present three confidentiality-preserving approaches to perform the recomputation: prover recomputation inside a verifier-controlled enclosure, verifier recomputation inside a prover-controlled enclosure, and zero-knowledge proofs. We describe the architecture, protocol, potential attacks, and sketch a plan to prototype this system.

1. Introduction

International agreements constraining AI development could reduce the competitive pressure to build increasingly dangerous systems, but such agreements require verification to be stable (Wasil et al., 2024; Baker et al., 2025; Harack et al., 2025; Scher & Thiergart, 2025). For example, nuclear testing agreements were enabled by seismometers (Baker, 2023), while bioweapons agreements have been limited by a lack of practical verification technology. Frontier AI development requires large quantities of specialized chips, which are produced by only a few companies worldwide (Sastry et al., 2024). This makes large concentrations of AI compute a practical target for verification (Shavit, 2023). The cost of such agreements could be significantly reduced by verification that allows declared workloads like inference to continue, while making restricted workloads like large-scale training impractical to run without detection. Hardware-enabled mechanisms such as trusted execution environments have been proposed to support compute verification (Aarne et al., 2024; Kulp et al., 2024; Petrie et al., 2025), but their security depends on trust in the chip's supply chain. We propose a protocol for this setting, in which a compute operator (the prover) demonstrates compliance to an external party (the verifier), without requiring either party to trust the other's hardware.

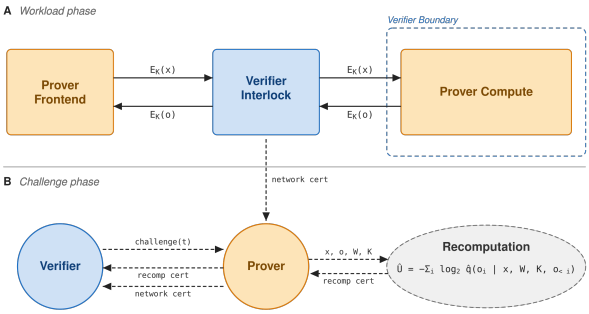

Figure 1. Two-phase protocol overview. (A) Workload phase: the prover frontend encrypts inputs as $E_K(x)$ and sends them through the verifier interlock to prover compute, which decrypts, computes the output o, and returns $E_K(o)$ through the verifier interlock to the prover frontend. The verifier interlock logs hashes of the encrypted traffic. (B) Challenge phase: the verifier issues a challenge, and the prover returns a recomputation certificate over (x, o, W, K) along with a network certificate from the verifier interlock. Together these attest to the unexplained information $\hat{U}$ for that output.

Specifically:

- The prover wants to use their compute node for legitimate work, and to protect the confidentiality and integrity of this workload. Dishonest provers want to perform covert workloads, but only if they will not be caught.
- The verifier wants to confirm that the prover node has very little useful bandwidth to extract results from undeclared workloads. Dishonest verifiers may want to steal or tamper with data, or disrupt prover systems.

[1]Future of Life Institute [2]Oxford Hardware AI Governance Lab. Correspondence to: James Petrie <james.petrie@protonmail.com>.

Accepted as a poster at the ICML 2026 Workshop on Technical AI Governance Research (TAIGR), Seoul, South Korea. Copyright 2026 by the author(s).

In this setting the prover and verifier may not trust each other's hardware because of concern about backdoors. Our proposal uses physical architecture and cryptography so that neither party needs to trust the other's hardware.

The idea is to isolate prover compute nodes so that all traffic passes through a channel monitored by the verifier. Traffic is encrypted so the verifier only records cryptographic commitments, not the data itself. The verifier periodically challenges the prover to demonstrate that randomly selected outputs are well-predicted by a policy-compliant computation on the measured inputs. Rather than requiring exact reproduction, the protocol measures how well a compliant computation can predict each output. This handles hardware nondeterminism naturally: the more predictable the outputs, the less bandwidth that could have been available for an undeclared workload.

We demonstrate this framework with inference as the primary use case. A viable method for verifying inference would enable agreements about AI development that do not disrupt commercially valuable inference, making such agreements substantially less costly. Rinberg et al. (2025) and Karvonen et al. (2025) address the unilateral setting with a trusted verifier, targeting steganographic weight exfiltration and inference-specification violations respectively. Our protocol extends this to the bilateral setting where neither the prover nor the verifier trusts the other's hardware.

We describe an architecture that could be built with existing technology to provide a high level of security. In this paper we:

- Define unexplained information and establish a general upper bound on it across output content and message timing.
- Describe a physical architecture that uses a verifier interlock to isolate the prover's compute node, and prover switches to encrypt traffic passing through this interlock.
- Detail three recomputation approaches and their respective trust assumptions.
- Apply this framework to inference, describing a protocol for performing confidential inference while maintaining credible logs and limiting timing information.
- Discuss potential attack vectors and defenses against them.
- Sketch initial plans for prototyping this system.

2. Protocol Overview

Each prover compute node is isolated from the world except for a single declared output channel, and potential covert side channels. In this paper we assume side channels would be infeasible to operate undetected at scale across thousands of nodes over several weeks (Cankaya et al., 2026b).

Component	Prover assumes	Verifier assumes
Prover Compute	Performs requested workload	Adversarial (could run arbitrary undeclared workloads)
Verifier Interlock	Adversarial (could attempt to read or modify traffic)	Signing key stays secret; executes commitment logic correctly
Verifier Boundary	Adversarial	Reliably detects high-bandwidth covert channels if attempted at scale across many nodes over weeks

The architecture is intended to be robust against nation-state adversaries operating in monitored facilities. Such actors may have significant resources, hardware expertise, and physical access to their own infrastructure. We therefore assume neither party trusts the other's hardware or supply chain. Dishonest provers may attempt to exfiltrate covert workload results through message contents, side channels, or metadata variations. Dishonest verifiers may attempt to steal model weights, tamper with computations, or disrupt prover systems.

To address these threats, the system isolates prover compute behind a verifier-controlled interlock, with encryption protecting prover data in transit. Figure 1 shows the full data path. The components and their trust assumptions are:

2.1. Workload Phase

During the workload phase, inputs x are encrypted with key K and the encrypted stream, $E_K(x)$, is sent to the prover compute via the verifier interlock, which logs a hash of the encrypted data. The prover compute generates a response stream, o, encrypts it with key K, and sends the encrypted output stream $E_K(o)$ through the verifier interlock, which again logs a hash. The prover compute logs x, o, and K. For inference workloads, this log is relatively small (accumulating at megabytes per second).

2.2. Challenge Phase

Occasionally, the verifier randomly selects a time, t, and challenges the prover to demonstrate that the output sent at t is well explained by a compliant computation. To do this, the prover collects:

- A network certificate from the verifier interlock, which includes a signature of the hash of the output data from time t, and the input data prior to time t (or a smaller subset of the input data, as discussed in the Inference Verification Protocol section).
- A recomputation certificate, which contains an attestation to the fraction of unexplained information when

comparing the measured output stream to the predictions from a compliant computation that only has access to the measured inputs (math for unexplained information is given in Bounding Unexplained Information, and security options for recomputation are analyzed in Confidential Recomputation).

The prover then sends the network certificate and recomputation certificate to the verifier. The verifier then checks that:

1. The cryptographic evidence in the certificates is genuine.
2. The commitments to input and output data are consistent between the network certificate and the recomputation certificate.
3. The network certificate is for the time that was challenged.

If all three of these are true, then the verifier now has a credible measurement of the amount of unexplained information in the output (expressed as a fraction, e.g. bytes unexplained / byte output). So long as the recomputation is confidential, then this protocol allows the verifier to make this measurement without learning any other information about the prover's workload.

3. Bounding Unexplained Information

The challenge phase requires the prover to demonstrate that outputs are well-explained by a compliant computation. In this section we formalize what "well-explained" means and establish an upper bound on the information available for covert payloads.

We model the declared channel as a discrete digital channel: sequences of tokens in, sequences of tokens out. The channel must be digital to ensure the entirety of the information is measured; any analog signals are treated as side channels.

Let x denote the measured inputs and o the measured outputs. The prover claims to be running a declared computation D that, given inputs x, produces outputs $o = D(x)$. The **unexplained information** is the information in the output not determined by $D(x)$. Any covert payload must be hidden in the output. The more of the output that is accounted for by $D(x)$, the less room remains.[1]

[1]Let M be any covert payload encoded in O. The prover is allowed to have $D(x)$, so only information in O beyond what $D(x)$ determines is available for covert communication. Formally, for $I(M; O \mid D(x))$ the Shannon mutual information between M and O conditional on $D(x)$, and $H(O \mid D(x))$ the entropy in O conditional on $D(x)$, then $I(M; O \mid D(x)) = H(O \mid D(x)) - H(O \mid M, D(x)) \leq H(O \mid D(x))$, since $H(O \mid M, D(x)) \geq 0$. This holds for any M.

If $D(x)$ were perfectly deterministic, the verifier could simply check whether $o = D(x)$. In practice, hardware nondeterminism (e.g., non-associativity of floating-point addition across different execution orders) means the same computation can produce slightly different outputs across runs. Let O denote the random variable over outputs the hardware could produce given $D(x)$, and o the output actually measured. The unexplained information is the conditional entropy:

$$U(O) = H(O \mid D(x)) = -\sum_{O} P(O \mid D(x)) \log_2 P(O \mid D(x))$$

where $H(O \mid D(x))$ denotes the conditional Shannon entropy, the expected number of bits needed to describe the output given $D(x)$.

The verifier may not be able to compute $U(O)$ exactly, since it depends on the true distribution P over hardware outcomes. However, for any valid probability distribution $Q(O \mid D(x))$, Gibbs' inequality guarantees:

$$U(O) = -\mathbb{E}_P[\log_2 P(O \mid D(x))] \leq -\mathbb{E}_P[\log_2 Q(O \mid D(x))]$$

A better estimate gives a tighter bound, but the bound is valid for any Q. Concretely, Q is the prediction of what outputs their hardware would produce, given the declared computation $D(x)$. This means the choice of Q can be delegated to the prover: a poor choice only increases the prover's own unexplained information, so the prover is incentivized to be as accurate as possible. They are also naturally the best source of an accurate Q, since they know the details of their own computation.

Factoring Q autoregressively as $Q(O \mid D(x)) = \prod_{i=1}^{n} Q_i(O_i \mid D(x), O_{<i})$ for tokens 1 to n, and evaluating on the measured output o gives the **estimated unexplained information**, the sum of per-token surprisals:

$$\hat{U}(o) = -\sum_{i=1}^{n} \log_2 Q_i(o_i \mid D(x), o_{<i})$$

This is the quantity attested to in the recomputation certificate. What constitutes a valid declared computation D, and how the recomputation is performed confidentially, is discussed in the next section.

4. Confidential Recomputation

The previous section established that unexplained information $U(O)$ can be bounded for any declared computation D. This section presents three approaches to performing the

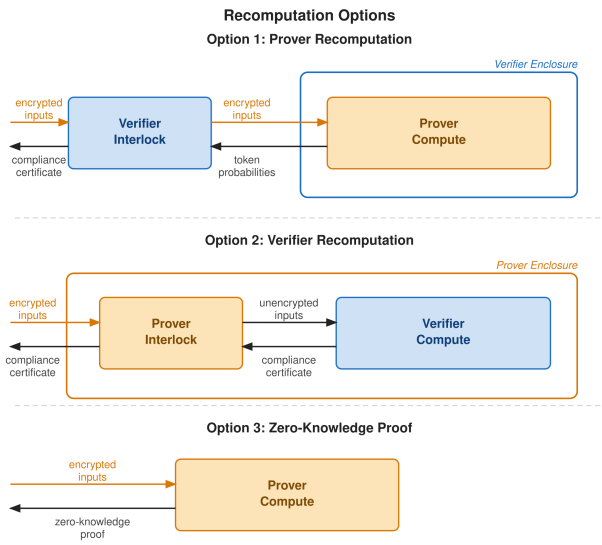

Figure 2. Recomputation Options: 1) prover compute inside a verifier enclosure with encrypted data provided by a verifier interlock, 2) verifier compute inside a prover enclosure with unencrypted data provided by a prover interlock, 3) zero-knowledge proof generated on prover compute.

recomputation confidentially (Figure 2), and examines what kinds of claims about D each approach can support.

The three recomputation approaches are:

- **Prover recomputation:** the prover's hardware performs the recomputation inside a verifier-controlled enclosure that prevents undeclared inputs from entering. The verifier interlock computes $\hat{U}(o)$ from the token probabilities without seeing decrypted data.
- **Verifier recomputation:** the verifier's hardware performs the recomputation inside a prover-controlled enclosure that prevents exfiltration of secrets. A prover interlock decrypts the data before passing it to verifier compute, and checks that the compliance certificate is exactly accurate before exporting it.
- **Zero-knowledge proof:** the prover generates a cryptographic proof of $D(x)$ and $\hat{U}(o)$ on their own hardware, with the tokens and model weights hidden but the computation graph visible to the verifier.

All three preserve the confidentiality of the prover's secrets without allowing the prover to tamper with the result. Option 1 supports only claims about what data entered the computation and how long it took. Options 2 and 3 additionally support claims about the computation itself, for example that a node is running inference with a specific model architecture. We compare these in detail in the Supported Claims about the Declared Workload section.

Component	Prover assumes	Verifier assumes
Prover Compute	Performs recomputation faithfully	Adversarial (could run arbitrary computation)
Verifier Enclosure	Adversarial	Prevents undeclared inputs from reaching prover compute
Verifier Interlock	Adversarial	Executes token-feeding and signing logic correctly; keeps signing key secret

4.1. Option 1: Prover Recomputation

For this option, the prover's own hardware performs the recomputation. The verifier's role is limited to controlling the enclosure and running the interlock that computes $\hat{U}(o)$.

The prover compute node sits inside a verifier-controlled enclosure that prevents any undeclared inputs from entering. The verifier interlock sits outside the enclosure and mediates all communication.

Only pre-committed information may enter the enclosure. This can be enforced by completing setup before the workload phase begins, or by cryptographically verifying that loaded data matches prior commitments.

When a challenge is issued:

1. The verifier interlock receives the encrypted inputs and the encrypted output for the challenged response.
2. The verifier interlock passes the encrypted inputs to the prover compute inside the enclosure.
3. The prover compute decrypts the inputs and runs the declared computation, producing predicted token probabilities $Q_i(o_i \mid D(x), o_{<i})$ for the next output position.
4. The prover compute outputs a probability distribution over the token vocabulary, indexed by encrypted token identifiers $E_K(token_j)$, so that the interlock can look up the probability assigned to the observed encrypted output token without learning which plaintext token it corresponds to.
5. The verifier interlock looks up the probability assigned to the encrypted output token, records the surprisal $-\log_2 Q_i(o_i \mid D(x), o_{<i})$, then reveals the encrypted token back to the prover compute.
6. Steps 3–5 repeat sequentially for each token, accumulating $\hat{U}(o)$.
7. The verifier interlock signs a compliance certificate attesting to $\hat{U}(o)$, o, and x.

The tokens are fed sequentially so that the prover commits probabilities for each token before seeing what it actu-

Component	Prover assumes	Verifier assumes
Verifier Compute	Adversarial (could attempt to exfiltrate data)	Performs recomputation faithfully
Prover Enclosure	Prevents exfiltration of secrets from verifier compute	Adversarial
Prover Interlock	Decrypts data and checks compliance certificate before releasing	Adversarial

Component	Prover assumes	Verifier assumes
Prover Hardware	Performs proof generation faithfully	Adversarial
Proof System	Correctly hides private inputs	Proof is sound (cannot be forged for a false statement)

ally was. The verifier need only trust the interlock, which performs simple token-feeding and surprisal computation. Because the probabilities are mapped to encrypted token indices under the encryption key K, the interlock sees the shape of each distribution but not which plaintext tokens the probabilities correspond to. If the prover wants to hide the shape of the distribution, they could use a fixed shape to make it zero knowledge, at the expense of less accurate predictions (e.g., always using 0.7, 0.1, 0.05 for the top three probabilities).

4.2. Option 2: Verifier Recomputation

In this option, the verifier's own hardware performs the recomputation. The verifier compute node sits inside a prover-controlled enclosure, which prevents side channels from exfiltrating secrets. The prover interlock sits at the boundary, decrypting inputs before they reach verifier compute and verifying that the compliance certificate produced by verifier compute does not contain any secret information before releasing it from the enclosure.

The verifier compute can directly verify that any data loaded into the enclosure (such as model weights) matches prior commitments before using it in the recomputation. To limit the attack surface from adversarial prover-supplied code, the verifier compute should run a fixed, audited execution engine that accepts a structured model description, such as a constrained directed acyclic graph of transformer blocks with declared dimensions and activations, rather than arbitrary executable code. As an additional defense, the prover may deploy multiple verification nodes with different hardware and software stacks in separate enclosures, with each challenge routed randomly to one of them, so that a dishonest verifier would need to compromise all variants to reliably exfiltrate information.

When a challenge is issued:

1. The prover interlock receives the encrypted inputs and encrypted output for the challenged response.
2. The prover interlock decrypts the data and passes the plaintext inputs to the verifier compute inside the enclosure.
3. The verifier compute runs the declared computation using sequential token-feeding (implemented in software, or optionally with dedicated hardware), producing predicted token probabilities $Q_i(o_i \mid D(x), o_{<i})$ and accumulating $\hat{U}(o)$.
4. The verifier compute signs a compliance certificate attesting to $\hat{U}(o)$.
5. The prover interlock independently computes the expected certificate contents and checks that the verifier's certificate matches exactly. It also checks that the signature matches the verifier's public key. Only if both checks pass does the prover interlock release the certificate from the enclosure.

The verifier compute sees plaintext data, but the prover-controlled enclosure prevents exfiltration. The compliance certificate is the only data that leaves the enclosure, and only after the prover interlock has verified that its contents exactly match what is expected. This ensures the certificate cannot encode hidden information.

4.3. Option 3: Zero-Knowledge Proof

In this option, the prover generates a cryptographic proof on their own hardware that the output is consistent with the declared computation, eliminating the need for physical enclosures entirely. The prover computes the proof locally and sends it to the verifier, who checks its validity.

The prover commits to model weights W via a cryptographic commitment scheme such as Pedersen commitments (Pedersen, 1991). The verifier receives only the commitment, not the weights themselves.

When a challenge is issued:

1. The prover collects the network certificate for the challenged output from the verifier interlock.
2. The prover generates a zero-knowledge proof that the output tokens, given the committed model weights and measured inputs, have estimated unexplained information $\hat{U}(o)$ as computed by the sum of per-token surprisals under $Q_i(o_i \mid D(x), o_{<i})$.
3. The prover sends the proof and network certificate to

the verifier.

4. The verifier checks that the proof is valid.

No re-execution or physical enclosure is needed to check for information leakage, since the proof is constructed to reveal nothing beyond $\hat{U}(o)$. The proof could also be made interactive, allowing the verifier to issue random challenges during proof generation, which can reduce proving costs at the expense of requiring real-time communication.

The proof reveals nothing beyond the validity of the statement: that the declared computation on the committed weights and measured inputs produces $\hat{U}(o)$ at most the claimed value. Model weights, inputs, and outputs remain private to the prover. Unlike the physical recomputation options, confidentiality does not depend on physical enclosures or supply chain integrity. Some ZKP implementations require that the model architecture (but not weights) be revealed to the verifier. With a more complex ZKP design, the model architecture could also be hidden by relying on automated checks for architecture properties within the proof itself.

The soundness of the entropy bound depends on the proof system leaving zero prover freedom in the intermediate values that determine Q_i. In the zkLLM protocol (Sun et al., 2024), the main sources of slack are the softmax shift (where multiple integer shifts can satisfy a tolerance band on the row sum), lookup table rounding (where the rule for discretizing real-valued entries is unspecified), and prover-chosen setup parameters. Each allows the prover to select among valid witnesses in a direction that inflates $Q_i(o_i)$ and depresses $\hat{U}(o)$ below the true value. Our companion paper closes these gaps by pinning setup parameters at the protocol level, fixing a deterministic rounding rule, and replacing each tolerance band with a bracket that pins the value to a unique integer, for instance pinning the softmax shift by monotonicity rather than by a tolerance on the row sum (Petrie, 2026).

Extending their protocol to prove the conditional entropy bound is inexpensive relative to the forward pass, because the bound is computed directly from the logits the forward pass already produces, using almost the same exponential and lookup operations as softmax. Our prototype models hardware nondeterminism as Gaussian noise on logits, giving token probabilities $q(o_i) \propto \Phi\left(\frac{\ell_i - v^*}{\sigma\sqrt{2}}\right)$ where v^* is the maximum logit, ℓ_i is the logit for token i, and σ is calibrated empirically; this extends to non-zero temperature sampling via the Gumbel max trick (Karvonen et al., 2025; Rinberg et al., 2025). The dominant cost is the forward pass itself. The largest prior zero-knowledge inference result, zkLLM (Sun et al., 2024), reached 13-billion parameters; our companion paper demonstrates an end-to-end proof of a 400-billion-parameter model on a single machine. For the proof system used in that demonstration, proving cost scales with the size of the committed intermediate states (the witness), which for long context lengths is dominated by the softmax proof over the attention intermediates (above roughly 11k tokens for Llama-4-Maverick). The companion paper's cost model projects several days of proof time for a one-million-token context on an NVL72-class cluster, which is feasible for this setting.[2]

The ZKP approach removes the need for physically enclosed recomputation hardware and provides the strongest protections for model weights, though current implementations reveal the model architecture to the verifier. The larger open problem is precision. Efficiently proving low-precision floating-point formats such as FP4 has not yet been demonstrated for the large matrix multiplications where we rely on Freivalds' check, so the committed integer model may diverge enough from the deployment to leave a substantial amount of unexplained information. Committing the low-precision floating-point intermediates directly, instead of an integer approximation, would remove this gap and is a plausible route to tighter bounds, though it needs further work.

4.4. Supported Claims about the Declared Workload

All three approaches can bind outputs to pre-committed data without the verifier seeing the plaintext. They differ in what additional claims about the computation the verifier can credibly enforce.

Option 1 (prover recomputation) supports only claims about what data entered the computation and how long it took. The verifier interlock sees nothing of the computation itself, so it cannot verify which algorithm or architecture was used, only that the output was produced from the pre-committed and measured inputs within a declared resource budget.

Option 2 (verifier recomputation) and **Option 3 (zero-knowledge proof)** additionally support detailed claims about the computation, such as specific architectures, specific decoding procedures, or the absence of intervening gradient updates. In both cases the allowed workloads must be fixed before the output is measured, so that the prover cannot retroactively choose a claim that fits whatever they actually ran. In Option 2, the prover interlock only releases

[2] At a one-million-token context the witness is of order 4×10^{16} slots (roughly $40320\,L^2$ for a context length of L tokens, almost entirely the softmax proof over the attention intermediates at that context). Proving cost is set by the two Reed-Solomon NTTs per slot, memory-bandwidth bound at roughly 0.42 ns per element on a single GB10. Because this floor rides aggregate memory bandwidth, an NVL72-class cluster provides on the order of 2,500 times a single GB10's bandwidth, bringing the projection from about twenty years on one machine to several days on the cluster.

a certificate when the verifier's recomputation matches a pre-agreed workload type, while in Option 3, the allowed workloads are compiled into a proof circuit committed to in advance.

These stronger claims are useful when policy needs to go beyond which pre-committed data is used. For example, Options 2 and 3 can verify that a specific computation was executed, while Option 1 relies on blunter constraints (such as limiting the amount of new data that can influence the computation) to make non-inference workloads impractical. Option 1 can be complemented by additional measures such as input bandwidth restrictions and memory wipes, discussed in concurrent work (Marks et al., 2026), that constrain covert workloads without requiring verification of the computation itself.

Option 1 is also substantially simpler to build, since the verifier's software is limited to token-feeding and surprisal computation. Option 2 requires the verifier to accept and execute prover-supplied workload descriptions, a significantly larger attack surface.

5. Inference Verification Protocol

In the Protocol Overview we treated the input and output as arbitrary discrete channels where each output token could depend on all prior inputs and outputs. In this section we specialize to the inference verification use case, where multiple users run inference workloads simultaneously and each output depends on only a fraction of prior inputs and outputs.

We present one concrete protocol that illustrates how the general framework could be applied to inference. The protocol is organized around request-response packets rather than persistent streams, so that recomputation only requires the referenced inputs rather than the entire prior stream. It introduces prover switches that handle encryption and protocol requirements so that the system can be retrofitted to existing prover datacenters with minimal disruption to the rest of the infrastructure.

5.1. Packet Format

Request (originates outside the compute node):

- **Request ID.** Unique identifier allowing responses to be paired with requests.
- **Request type.** Either inference or non-inference, determining how the response is accounted for.
- **References.** List of IDs of previous requests and responses whose contents are needed to assemble the full input context for this computation (e.g., prior turns in a conversation).
- **Payload.** Request content (e.g., input tokens for inference, or a telemetry query for non-inference).
- **Cryptographic commitment.** Hash of the symmetric key and nonces used to encrypt the payload and the corresponding response, allowing the verifier to later confirm that the encryption was consistent.
- **Predicted response size distribution.** The prover's committed distribution over the byte length of the response.
- **Predicted response time distribution.** The prover's committed distribution over response arrival time, discretized into timing buckets.

All fields except the predicted distributions are encrypted. The predicted distributions are sent in plaintext so that the interlock can record them without needing decryption capability. They reveal expected latency and response length but nothing about the content of the computation. Because requests originate outside the compute node, their contents are inputs to the computation and do not need to be accounted for as unexplained information.

Response:

- **Request ID.** Matching the originating request.
- **Payload.** Response content (e.g., output tokens).

Both packet types also include a payload length field on the wire. Because responses originate inside the compute node, all of their contents must be accounted for. The request ID is determined by the request, so it is counted as zero unexplained information. The payload content is accounted for via the challenge protocol for inference responses, or counted at full size for non-inference responses. The same bound applies to response timing and size: the surprisal of the observed values under the distributions committed in the request is added to $\hat{U}(o)$ for that response.

Each packet is treated independently for accounting purposes. Any statistical dependence between packets would only reduce unexplained information, so independence is a valid assumption for an upper bound.

5.2. Components

The prover inference cluster, verifier interlock, and verifier inference boundary are as described in the protocol overview. This protocol introduces prover switches before and after the verifier interlock, allowing the rest of the prover system to treat the switch, interlock, switch path as a single component that repeats messages with minor additional latency. Dedicated prover switches are not strictly required; other hardware could be repurposed to act as them.

Prover Switches. A pair of switches sit on either side of the verifier interlock, sharing an ephemeral symmetric key. They encrypt and decrypt traffic passing through the

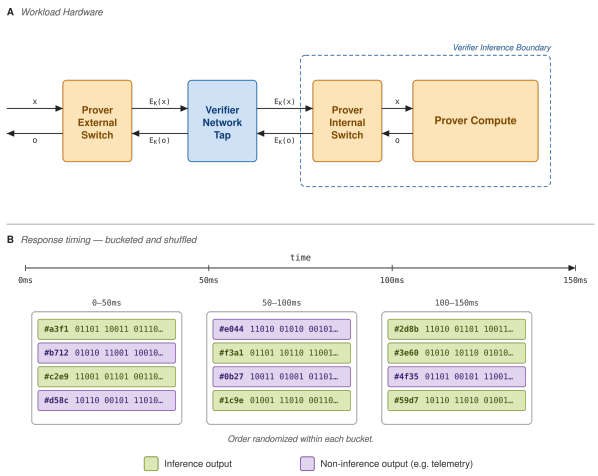

Figure 3. (A) Hardware setup: prover switches on both sides of the verifier interlock. (B) Response timing: responses are grouped into fixed time buckets and shuffled within each bucket so that only bucket membership is observable.

verifier interlock, and handle the prover side of the request-response protocol: key management, response time and size estimation, and encryption. The internal switch aggregates traffic from multiple internal network cables into a single outgoing cable. The external switch stores messages sent and received for responding to verification challenges.

In addition to its role described in the protocol overview, the interlock for this protocol records the predicted distributions from each request, the timestamp of each response, and collects responses into fixed time buckets (e.g., 10ms) and randomizes their order within each bucket, so that only bucket membership is observable. If storage is a concern, the interlock can record hashes of the predicted distributions rather than the distributions themselves, and verify them against the plaintext at challenge time.

5.3. Unexplained Information Budget

The protocol allows the prover flexibility in how deterministic to make their computation, how precise to make their timing predictions (smaller time buckets require more precise predictions, and imprecise predictions increase unexplained information), and how much non-inference traffic to permit. Rather than the verifier being prescriptive about how the prover manages these tradeoffs, the verifier can set a total budget for the unexplained information rate. The prover then optimizes their own hardware, compute usage, and protocols to stay within this budget. The prover is better placed to do this as the developer of the hardware and software, and because some of the relevant details may be confidential.

6. Discussion

The protocol achieves its verification guarantees through successive layers of bandwidth reduction. For the inference use case developed in this paper, physical isolation reduces output bandwidth from TB/s to GB/s by removing undeclared network connections; the verifier interlock further restricts the declared channel to MB/s; and recomputation bounds the unexplained fraction of that remaining bandwidth to KB/s or less.

In principle, unexplained information can be pushed close to zero. Deterministic GPU kernels have been demonstrated in practice (DeepSeek-AI, 2026; Cankaya et al., 2026a), eliminating hardware nondeterminism as a source of unexplained information in output content. Reaching near-zero overall would additionally require fixed response sizes and timing, for example by padding responses to a fixed size and having prover compute always reply within a fixed time window, with the external switch issuing a follow-up request if the response is not yet complete. Hardware status updates would also need to be minimal, perhaps a single all-ok heartbeat bit per second.

Recomputation option 2 requires expanding the prover's trust boundary to include the verifier's compute within a prover enclosure, which may be a burden for ultra-high-security applications. Options 1 and 3 do not have this limitation, since the recomputation happens on the prover's own compute, already within the trust boundary. Even where full confidentiality cannot be guaranteed, the protocol could still be useful for verifying inference of less sensitive models, where the cost of a potential confidentiality breach is outweighed by the benefit of credible verification.

The architecture also depends on a physical assumption: that sustained high-bandwidth covert side channels cannot go undetected across thousands of nodes over weeks (Cankaya et al., 2026b).

Quantum computers capable of breaking classical cryptographic assumptions may arrive in the near future, potentially accelerated by transformative AI. Signatures used by the interlock, and by verifier compute in Option 2, can use post-quantum schemes such as SPHINCS+ from the start, since the system need not optimize for signature size or speed. The zero-knowledge proof option presents a harder challenge: current implementations depend on the discrete logarithm assumption for Pedersen commitments. Zero-knowledge proofs can be implemented without this assumption, for example using FRI-based commitment schemes, but these hash-based commitments lose the homomorphism properties that Pedersen commitments provide, and efficiently replacing them is an active area of research.

The main costs are computational overhead and the difficulty of expressing AI or hardware architectures as a proof circuit.

This is a software problem, and one unusually suited to AI-assisted development: because soundness rests only on a small verifier and the proof system, the prover's code can be optimized freely, including by AI, without needing to be trusted or reviewed, since any error can only cause a proof to fail rather than to falsely verify.

This paper uses inference as its primary example, but the same protocol could be used as a tool within broader compute verification schemes. The interlock's role as the sole network path to an isolated compute node supports verification of other resource dimensions, such as ingress, memory, and compute usage (Marks et al., 2026), and could be extended to training workloads by tapping inter-node links within a cluster, if its throughput and hashing rate were scaled to match training bandwidth (Peigne-Lefebvre et al., 2026).

The protocol also has unilateral applications. Model developers could use it to protect weights from exfiltration (Rinberg et al., 2025; Karvonen et al., 2025). Auditors could verify that a deployed model matches the version that was tested and approved. And operators of high-stakes AI workloads could use the cryptographic evidence to detect bugs or sabotage.

References

Aarne, O., Fist, T., and Withers, C. Secure, governable chips: Using on-chip mechanisms to manage national security risks from AI & advanced computing. Technical report, Center for a New American Security, 2024.

Baker, M. Nuclear arms control verification and lessons for AI treaties. arXiv:2304.04123, 2023.

Baker, M., Kulp, G., Marks, O., Brundage, M., and Heim, L. Verifying international agreements on AI: Six layers of verification for rules on large-scale AI development and deployment. Technical Report WR-A4077-1, RAND Corporation, 2025.

Cankaya, N. et al. Bit-exact AI inference verification without performance tradeoffs, 2026a. Forthcoming.

Cankaya, N. et al. Suppressing side channels in an untrusted datacenter via retrofitted defenses, 2026b. Forthcoming.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026.

Harack, B., Trager, R. F., Reuel, A., Manheim, D., Brundage, M., Aarne, O., Scher, A., Pan, Y., Xiao, J., Loke, K., Adan, S. N., Bas, G., Caputo, N. A., Morse, J. C., Ahuja, J., Duan, I., Egan, J., Bucknall, B., Rosen, B., Araujo, R., Boulanin, V., Lall, R., Barez, F., Alvira, S., Katzke, C., Atamli, A., and Awad, A. Verification for international AI governance. Technical report, Oxford Martin AI Governance Initiative, 2025.

Karvonen, A., Reuter, D., Rinberg, R., Marks, L., Garriga-Alonso, A., and Warr, K. DiFR: Inference verification despite nondeterminism. arXiv:2511.20621, 2025.

Kulp, G., Gonzales, D., Smith, E., Heim, L., Puri, P., Vermeer, M. J. D., and Winkelman, Z. Hardware-enabled governance mechanisms: Developing technical solutions to exempt items otherwise classified under export control classification numbers 3A090 and 4A090. Technical report, RAND Corporation, 2024.

Marks, L., Reuter, D., Ng, J., Carlucci, A., Baker, M., Petrie, J., Hausenloy, J., and Karvonen, A. Verifiable computational accounting for AI developers. In *ICML Workshop on Technical AI Governance Research*, 2026. URL https://openreview.net/forum?id=ajrtHuFw2S.

Pedersen, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pp. 129–140. Springer, 1991. doi: 10.1007/3-540-46766-1_9.

Peigne-Lefebvre, P., Nguyen, K., and Wang, P. Zero knowledge verification for frontier AI training is possible, 2026. Forthcoming.

Petrie, J. VerInf: Zero-knowledge proofs of frontier-scale LLM inference by bounding unexplained information, 2026. Working title; manuscript in preparation.

Petrie, J., Aarne, O., Ammann, N., and Dalrymple, D. Flexible hardware-enabled guarantees (flexHEG). arXiv:2506.15093, 2025.

Rinberg, R., Karvonen, A., Hoover, A., Reuter, D., and Warr, K. Verifying LLM inference to detect model weight exfiltration. arXiv:2511.02620, 2025.

Sastry, G., Heim, L., Belfield, H., Anderljung, M., Brundage, M., Hazell, J., O'Keefe, C., Hadfield, G. K., Ngo, R., Pilz, K., Gor, G., Bluemke, E., Shoker, S., Egan, J., Trager, R. F., Avin, S., Weller, A., Bengio, Y., and Coyle, D. Computing power and the governance of artificial intelligence. arXiv:2402.08797, 2024.

Scher, A. and Thiergart, L. Mechanisms to verify international agreements about AI development. arXiv:2506.15867, 2025.

Shavit, Y. What does it take to catch a chinchilla? Verifying rules on large-scale neural network training. arXiv:2303.11341, 2023.

Sun, H., Li, J., and Zhang, H. zkLLM: Zero knowledge proofs for large language models. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*. ACM, 2024.

Wasil, A. R., Reed, T., Miller, J. W., and Barnett, P. Verification methods for international AI agreements. arXiv:2408.16074, 2024.

7. Appendix A: Attacks and Countermeasures

This verification architecture aims to defend against nation-state actors attempting to covertly circumvent an international agreement. Such actors may have significant resources, hardware expertise, and physical access to their own infrastructure, but are assumed to prefer covert circumvention over overt violation that would be diplomatically costly.

7.1. Prover Attacks

The prover wants to perform noncompliant workloads at scale without detection.

Attack Vector	Description	Countermeasures
Side channels from workload cluster	Exfiltration via radio, power lines, undeclared cables, lasers, etc.	Physical inspections for undeclared communication paths; active monitoring for anomalous signals; full disassembly of a random sample of clusters.
Forged network commitments	Prover compromises the verifier interlock's signing key and forges network certificates for fabricated traffic.	Interlock uses tamper-responsive key deletion and side channel hardening (see Appendix B). Merkle chain over past hashes forces any forgery to maintain consistent state from the point of compromise.
Information in metadata or telemetry	Packet headers, sensor telemetry, or non-inference traffic encode covert information.	All non-inference data counts at full entropy against the prover's unexplained information budget. Interlock pauses or flags communication if the budget is exceeded.
Information in response timing	Timing of response packets encodes covert information.	Interlock randomizes response order within fixed time buckets. Deviations from the predicted response time distribution committed in the request are counted against the entropy budget.
Training disguised as inference	Gradient updates or other training data are sent as inference requests, with outputs carrying training results.	Options 2 and 3 verify the computation graph, which can enforce that only inference (not backpropagation) was performed. Option 1 cannot distinguish architectures, so it requires blunter restrictions such as input rate caps to limit the bandwidth available for receiving gradient updates from other nodes.
Manipulating recomputation (Options 1, 3)	In Options 1 and 3, the prover runs the recomputation and could attempt to inflate $Q_i(o_i)$ to deflate $\hat{U}(o)$.	Option 1: sequential token-feeding prevents the prover from seeing a token before committing probabilities. Option 3: proof system must eliminate prover degrees of freedom (see Option 3 discussion of slack).

7.2. Verifier Attacks

The verifier wants to steal model weights, algorithmic insights, or inference transcripts, or to tamper with or sabotage prover operations.

Attack Vector	Description	Countermeasures
Side channels from recomputation enclosure	Verifier extracts confidential data from inside the prover-controlled enclosure via electromagnetic, acoustic, or optical channels.	SCIF-equivalent secure enclosure; prover interlock mediates all exported data. Note that highly compressible algorithmic insights (e.g., a single architecture choice) may be harder to protect against low-bandwidth leakage.
Information in verification certificates	Certificate contents or signature nonces encode covert information.	Prover interlock independently recomputes expected certificate contents and checks for an exact match before release. Signature nonces are derived deterministically from the message so the prover can verify them without the signing key.
Information in verification errors	Recomputation server selectively fails to produce correct certificates, encoding one bit per challenge (e.g., correct certificate for "no," incorrect for "yes," with the question encoded in challenge timing).	Prover could substitute equivalent but less efficient model descriptions that omit proprietary details. Verifier must provide a credible explanation for any failure.
Tampering with inference responses	Verifier extracts the prover switch key via side channel (switches are physically adjacent to the interlock) and uses it to decrypt, modify, and re-encrypt inference traffic.	Side channel hardening on prover switches (see Appendix B). Prover can run independent spot-checks of their own inference results to detect modifications.

8. Appendix B: Inference Verification Hardware

This appendix describes hardware requirements, a candidate design, and a prototype plan for the inference verification protocol described in the Inference Verification Protocol section.

8.1. Verifier Interlock Requirements

The interlock must perform a small number of operations at line rate (hashing, timestamping, bucket randomization) while resisting both compromise and side channel leakage. It operates in a physically adversarial environment: the prover may have sophisticated equipment making nearby measurements.

Throughput.

- Continuous forwarding of at least approximately 100 MB/s input and 100 MB/s output.
- Hashing at approximately 200 MB/s or above to keep pace with bidirectional traffic.

Packet order randomization. The interlock collects responses into fixed time buckets and randomizes their order within each bucket before forwarding. This requires buffering up to one bucket's worth of responses and performing a uniform random shuffle before releasing the batch. The buffer size and shuffle latency scale with the bucket duration and traffic rate; for a 10ms bucket at 100 MB/s output, the buffer is on the order of 1 MB.

Cryptographic signing. The interlock must sign network certificates. Two options:

- Public key signatures allow the prover to independently verify that certificates are well formed, confirming they do not encode hidden information. This is more complex cryptographically.
- Pre-shared symmetric MACs are simpler but prevent the prover from checking certificate contents, since the prover cannot distinguish a valid MAC from one that encodes a covert message.

Post-quantum signature schemes should be used regardless of the choice. Interlock hardware may be deployed for years or decades, a timeline that may overlap with the arrival of practical quantum attacks on classical signature schemes.

Side channel hardening. The prover may attempt to extract the interlock's signing key via timing, power, or electromagnetic side channels. The interlock should use constant time cryptographic implementations and, where possible, power filtering

or isolation on external connections (power cables, cooling tubes). An optical data diode on the output path can prevent electrical signals from propagating back into the interlock, defending against voltage glitches intended to disrupt its operation or induce exploitable faults.

Physical protections (nice to have).

- Tamper resistance or tamper evidence on the enclosure.
- Tamper responsive deletion of signing keys.
- Dedicated key storage (e.g., secure element), or keys held in volatile memory with battery backup so they are never written to persistent storage.

Logical interface. Communication between the prover switches and verifier interlock uses a standard framing protocol (e.g., Ethernet). The interlock sanitizes incoming traffic, stripping framing metadata and reframing for the other side, ensuring that framing itself cannot carry information. The only essential requirement is that the interlock maintain a single internal channel that operates purely in the specified packet format, with no additional information such as protocol headers. This interface is a good candidate for formal verification.

As an additional security measure, the interlock may include physical restrictions on absolute output bandwidth (e.g., transceivers designed to cap data rate), providing a hardware enforced ceiling independent of the protocol.

8.2. Prover Switch Requirements

The prover switches must encrypt and decrypt at line rate while protecting key material from the nearby interlock.

- Encrypt and decrypt at line rate using an ephemeral symmetric key (e.g., AES-256).
- The external switch stores the entirety of messages sent and received, retained for a configurable period (e.g., 30 days) sufficient to cover the challenge window.
- Side channel protections against key leakage, since the verifier interlock sits physically adjacent and could incorporate measurement equipment.

8.3. Candidate Production Design

A plausible production design for the verifier interlock:

- A commercial off the shelf (COTS) FPGA designed for security applications (e.g., a military grade FPGA with built in side channel countermeasures).
- Standard Ethernet input and output ports.
- Formally verified logic implementing: strip incoming frames to the specified packet format, forward over the internal interface, hash the contents, record the hash and timestamp, buffer and shuffle responses within each time bucket, then reframe for the outgoing port.

The simplicity of the interlock's logic (hash, timestamp, sign, shuffle, forward) makes formal verification tractable. The entire data path can be expressed as a small state machine with no branching on packet contents.

Prover switches have less stringent security requirements and could be implemented using reprogrammable network switches or SmartNICs with custom firmware.

8.4. Prototype Plan

The goal of an initial prototype is to demonstrate the end to end protocol on real hardware at a moderate security level, measuring latency overhead, unexplained information rates, and recomputation times.

Workload prototype:

Role	Hardware
Prover compute (inference)	GPU or DGX Spark
Prover external switch + simulated customer	CPU
Prover internal switch	Reprogrammable network switch or SmartNIC
Verifier interlock	FPGA or SmartNIC with custom firmware

The verifier interlock should implement the full packet processing pipeline (strip, hash, timestamp, shuffle, reframe) in hardware or gateware, providing a realistic test of the logical interface. An FPGA is preferred over a general purpose CPU for this role because it more closely approximates the production design and makes it easier to reason about the absence of unintended information paths through the device. The FPGA gateware developed for the prototype could also serve as the basis for a follow up formally verified implementation.

Recomputation prototype (Option 1: prover recomputation):

Role	Hardware
Prover recomputation compute	GPU
Verifier recomputation interlock	FPGA
Verifier enclosure (optional)	Small Sensitive Compartmented Information Facility (SCIF) or equivalent secure enclosure

Recomputation does not need to run at inference speed, since challenges can be issued with a delay. This means a single GPU with a large SSD may be sufficient, reducing both cost and the physical size of the security enclosure. For example, a 1 TB model can be streamed from SSD in approximately 16 seconds at 64 GB/s. Recomputing $L = 5{,}000$ tokens requires approximately $2NL$ FLOP, or 12 seconds on a single H100 at 800 TFLOP/s. With weight streaming and computation overlapping, the total is approximately 20 seconds.

The workload prototype would validate the packet format, encryption, timestamping, and bucket randomization. The recomputation prototype would validate the challenge flow, sequential token feeding, and surprisal computation. Combining both would allow end to end measurement of unexplained information rates under realistic inference workloads.