

Towards Verifiable AI with Lightweight Cryptographic Proofs of Inference[★]

Pranay Anchuri¹, Matteo Campanelli^{1,2}, Paul Cesaretti³,
Rosario Gennaro^{1,3,4}, Tushar M. Jois^{3,4}, Hasan S. Kayman⁴ and
Tugce Ozdemir³

¹ Offchain Labs

² University of Tartu, Estonia

³ CUNY Graduate Center

⁴ City College of New York

Abstract. When large AI models are deployed as cloud-based services, clients have no guarantee that responses are correct or were produced by the intended model. Rerunning inference locally is infeasible for large models, and existing cryptographic proof systems—while providing strong correctness guarantees—introduce prohibitive prover overhead (e.g., hundreds of seconds per query for billion-parameter models). We present a verification framework and protocol that replaces full cryptographic proofs with a lightweight, sampling-based approach grounded in statistical properties of neural networks. We formalize the conditions under which trace separation between functionally dissimilar models can be leveraged to argue the security of verifiable inference protocols. The prover commits to the execution trace of inference via Merkle-tree-based vector commitments and opens only a small number of entries along randomly sampled paths from output to input. This yields a protocol that trades soundness for efficiency, a tradeoff well-suited to auditing, large-scale deployment settings where repeated queries amplify detection probability, and scenarios with rationally incentivized provers who face penalties upon detection. Our approach reduces proving times by several orders of magnitude compared to state-of-the-art cryptographic proof systems, going from the order of minutes to the order of milliseconds, with moderately larger proofs. Experiments on ResNet-18 classifiers and Llama-2-7B confirm that common architectures exhibit the statistical properties our protocol requires, and that natural adversarial strategies (gradient-descent reconstruction, inverse transforms, logit swapping) fail to produce traces that evade detection. We additionally present a protocol in the refereed delegation model, where two competing servers enable correct output identification in a logarithmic number of rounds.

1 Introduction

The landscape of modern computing is dominated by deep neural networks which have seen tremendous success in a wide range of problem domains from biology and protein folding [JEP⁺21], to image classification and object arrangement in complex scenes with

E-mail: panchuri@offchainlabs.com (Pranay Anchuri), binarywhalesinternaryseas@gmail.com (Matteo Campanelli), pcesaretti@gradcenter.cuny.edu (Paul Cesaretti), rosario@ccny.cuny.edu (Rosario Gennaro), tjois@ccny.cuny.edu (Tushar M. Jois), hkayman@ccny.cuny.edu (Hasan S. Kayman), tozdemir@ccny.cuny.edu (Tugce Ozdemir)

[★]An earlier version of this work is to appear in the proceedings of IEEE SaTML 2026 [ACC⁺26]. This document incorporates minor revisions.

computer vision [KSH12, GDDM14] culminating in complex general-purpose tasks via large language models [ea25b, ea25a, ea24]. Due to their ubiquitous use as an external service via cloud-based platforms or APIs, there is urgency in developing protocols and methods to help safeguard this interface. In practice a client of these models will only receive an answer from an untrusted, possibly corrupted, server who could provide incorrect responses. A naive solution to this problem would be to allow the user to have complete access to the underlying model by downloading it from a verified site and executing the query on the user’s home device. This is clearly infeasible as the required computing resources for these networks are too large for the average user’s systems. The question, therefore, is:

Can we efficiently verify the computation of a large AI model performed by an untrusted server with less resources than what is needed to actually run them?

Most proposals to build verifiable AI systems use Cryptographic Proofs, in particular *Succinct Non-interactive Arguments of Knowledge* (or SNARKs) [BCCT12, GGPR13, PHGR16, Gro16] (see, e.g., [WH22, WYX⁺21, KHSS22, FQZ⁺21, LKKO24, LXZ21, WYX⁺21, QSL⁺25, SLZ24a, Sin24]). SNARKs allow the client to verify the correctness of an arbitrary computation from a short proof, achieving remarkably efficient verification (e.g., constant-size proofs and verification time, regardless of the size of the computation being proven).

The challenge with using SNARKs, however, is that they incur a large overhead for proof generation, typically 1000× or larger [Tha22]. The reason is that they introduce expensive cryptographic operations for *every* elementary computation step (for example, a gate in a circuit representation). In practice this leads to unacceptable delays in computing proofs: consider zkLLM [SLZ24b], where producing a proof of correct inference for a large language model with up to 13B parameters takes around 15 minutes. This is particularly problematic since verifiable inferences must be done at almost zero overhead, as predictions are computed many times and they are usually very fast to produce, so the proof computation has to be comparably fast¹. So, while work is ongoing to reduce the burden of using cryptographic approaches, full proofs on machine learning models using them still appear to be infeasible.

Further motivation. We briefly point out that efficient proofs for inference can be used to solve many other AI related security questions beyond outsourced computation. One of particular interest is *model transparency*: the ability to prove that an AI system—one that has been “certified” as having been trained correctly over certain “legal” data—is indeed the one being used in the field to compute predictions. Another is *fairwashing* in which a provider uses a biased model for users but substitutes an unbiased one when under audit [AAF⁺19]. See also Appendix A.

1.1 Our Contributions

We present a verification framework and protocols for *Verifiable Inference* of large AI models. **We take a lightweight approach to cryptographic proofs of inference and as a result are several orders of magnitude faster than previous protocols (e.g., milliseconds rather than minutes).** Our technical approach has not been previously used in this setting and is of independent interest.

A formal framework for verifiable inference from trace separation: One of our main conceptual contributions is a theoretical framework that can be used by protocol designers to leverage statistical properties of computation traces to argue the security

¹This may not necessarily hold for a scheme for verifiable *training*: training is a much more computationally intensive process, performed infrequently, and can tolerate slower proof production times. In our work we exclusively focus on proofs of inference, from an already correctly trained model.

of verifiable inference protocols. Our starting observation, grounded in results in the AI literature on representational similarity, is that two models that produce sufficiently different outputs must also differ in their internal activation traces. We formalize this as a *trace separation* property and show that, combined with an appropriate testing mechanism, it is sufficient to guarantee security (Section 4). Our framework is directly applicable to a relatively limited —though still of practical interest— security notion (which we call *other-model soundness*), under which the client can detect whether the server used a model whose behavior differs substantially from the intended one.

A concrete protocol and its experimental evaluation: We instantiate our framework with a concrete scheme that detects cheating via a small number of checks on random parts of the entire computation (in contrast to Cryptographic Proofs that check the correctness of every step yielding a proof of exact, rather than approximate, correctness). Key advantages of our construction are both simplicity and efficiency: the prover simply commits to the execution trace of the computation and then opens a small number of entries². This enables our improvement in proving time, of several orders of magnitude, compared to prior solutions.

We extensively validate our protocol’s assumptions and performance through a series of experiments. We show that common neural network architectures – classifiers and LLMs – have the statistical properties necessary to enable the testing of the correctness of the computation. We also investigate to which extent our techniques provide security in a stronger adversarial setting where cheating may follow *arbitrary* strategies, rather than merely substituting a different model, and show that our test approach is resistant to natural attacks of this kind. Additionally, we provide a performance comparison on commodity hardware and highlight our significant efficiency gains over prior work. An intuitive explanation of our protocol is presented in Section 2, with the full protocol given in Section 5.

A protocol in the refereed model: As an additional contribution, we present a protocol in the *refereed model* [CRR13], which assumes that two servers compute the inference and we are guaranteed that at least one does so honestly. If the servers make competing different claims, we present an efficient protocol to determine which one is correct which requires only a logarithmic (in the size of the model) number of steps. This can be found in Appendix D.

1.2 Related Work

Having discussed cryptographic proof approaches in the Introduction, we focus here on alternative methods. The recent work in [SLZ⁺25] introduced SVIP, a verifiable inference protocol that works by training a secret-dependent proxy task. A trusted intermediary generates and distributes secrets to users. However, this approach for verifiable inference has notable overheads: i) it requires model-specific training that can be resource-intensive. ii) The protocol’s security relies on active refresh of secrets to defend against secret recovery attacks, in which the provider attempts to steal the secrets by posing as a user and exploiting the proxy task. In contrast, our protocol provides a more general verification method. It operates directly between a prover and a verifier and does not depend on an intermediary to manage secrets. Furthermore, our protocol directly verifies the integrity of the inference process without requiring proxy tasks. This allows our protocol to naturally apply to a wide class of general deep neural networks beyond just LLMs.

AI Watermarking (see [ZGC⁺25b] for a survey) embeds certain signals into the output of an AI model to identify that content as AI generated, through later detection. These

²One consequence of this “cleartext opening” of values is that our protocol is not *zero-knowledge* and reveals information about the underlying model, still preserving our goal of enforcing correctness. If model confidentiality is an additional goal, in Section 9 we discuss possible ways to add ZK to our protocol, that should still perform much better than current state of the art protocols.

techniques solve a different problem than the one we address: they verify that *some* AI model was used, while our goal is to verify that a specific model was computed correctly.

Our main protocol can be seen as an example of *proof of proximity* introduced in [RVW13]. Most of the literature for proofs of proximity is focused on increasing the efficiency of the Verifier (e.g., by allowing proof verification without even reading the entire input, but accessing only a sublinear portion of it). We, instead, focus on reducing the prover overhead.

Our protocol in the refereed model follows the basic bisection blueprint first proposed in [CRR13]. This paradigm has been successfully adopted by so-called optimistic rollup protocols to scale executions on the Ethereum blockchain (e.g., Arbitrum [KGC⁺18], Optimism [Arm21]). The application of the refereed model to verify the outputs of large-scale AI systems was initially proposed in [CSYW24]. While that work correctly identifies the inherent suitability of the refereed computation model for this domain—given the sublinear verification complexity and the often-extensive computational traces of large AI models—it does not provide empirical results or an experimental validation of this hypothesis, which we do.

1.3 Outline

The rest of the paper is organized as follows. Section 2 provides an intuitive technical overview of our approach, including the key ideas behind our construction and the security notions we consider. Section 3 introduces preliminaries and notation. In Section 4 we present a formal framework for verifiable inference, including our security definitions, a theoretical treatment linking machine learning properties to soundness, and a generic compiler from idealized to cryptographic constructions using vector commitments. Section 5 describes our specific construction and discusses its security properties. Section 6 and Section 7 present our experimental evaluation. Section 8 reports on our performance evaluation and a comparison with SNARK-based approaches to verifiable inference. We conclude in Section 9.

2 An Intuitive Technical Overview

A neural network is a directed acyclic graph (DAG), where nodes represent neurons and edges are the weighted connections between them. The network is configured by a training process that yields a vector of weights $(w_{ij})_{ij}$ assigned to the directed edge connecting neuron i to neuron j . The state of each neuron is represented by a scalar value known as its activation. During evaluation, the input data is encoded as the activation values of the network’s source nodes. The activation of a subsequent neuron j is determined by a function of the weights and the activation values of its parent neurons. The standard formulation for this process is:

$$a_j = \phi\left(\sum_{i \in G_j} w_{ij} a_i\right)$$

where a_j is the activation of neuron j , G_j is the set of indices of its parent neurons, w_{ij} is the weight on the edge from parent neuron i to neuron j , and ϕ is a non-linear activation function.³ The final output of the neural network is represented by the activation values of the terminal neurons in the DAG. We refer to the list $(a_1, a_2, \dots)$ of activation values as the *execution trace* of the model (or simply as trace).

³Formulations often include a bias term b_i added to $w_{ij}a_i$, but for ease of vectorization we consider b_i an additional weight with an activation of 1. Also, neural networks can use different ϕ , but we fix it here for simplicity.

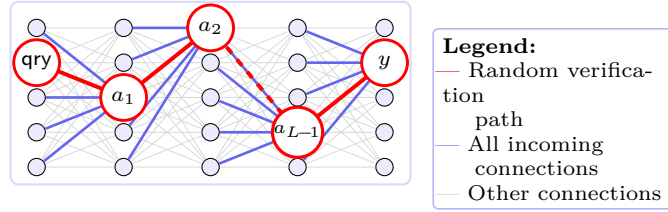


Figure 1: Path-based verification protocol. The prover commits to the full output vector for query $\mathbf{qry}$. Then, the verifier samples one path, starting from a single output neuron, picks one random neuron per layer (red path) and checks each selected neuron’s activation using all incoming connections (blue) from parent neurons in the previous layer. At each step, the new sampled neuron is picked among the parents. Counterfeit outputs lead to detectable inconsistencies with high probability.

Our methodology—simpler tests from similarity. As stated earlier, our goal is to design an *efficient* verification scheme for neural network inference, even for large AI models. In order to avoid the massive prover overhead of cryptographic proofs, we need to completely depart from their usual approach, which encodes the correctness of each step of the computation in an algebraic “circuit-checking” engine [Nit20].

Our starting point was the body of work in the AI research literature that has developed metrics to compare neural network representations (see [KSSL25] for a comprehensive survey). The idea of comparing internal representations originated in neuroscience, where pairwise dissimilarity responses of brain activity across various stimuli are used to build structured comparisons [KMB08], and this technique was later translated into AI to compare internal representations of deep neural network structures [KNLH19]. One of the motivations of this research is *Explainable AI*, i.e., the desire to understand what happens during the execution of a neural network that allows it to “understand” or “generate knowledge”.

Of particular interest to us were two notions of neural network similarity: *functional similarity* and *representational similarity*. Functional similarity measures the difference in the probability distributions of the outputs, while representational similarity focuses on quantifying similarity in the distributions of the internal activations. Importantly, as Klabunde et al. note [KSSL25], “When functional similarity measures indicate dissimilarity, representations must be dissimilar at some layer.” In other words, two models that produce sufficiently different output distributions must be different also in the distributions of their internal activations.

This suggests an intuitive way to test the correctness of a claimed output. Assume a client asks an untrusted server to compute a model $\mathcal{M}_1$ on input $\mathbf{qry}$ and the server responds with y . If the server is lying by using a substantially different AI model $\mathcal{M}_2$ the client can hopefully detect this by testing the two internal representations on a small number of samples. While exploring the literature on similarity measures, however, we could not find a similarity notion which would be suitable for this task (i.e., which guarantees that if the two internal distributions are sufficiently far apart then we can detect it from a few samples). We therefore developed our own approach.

Towards a construction. Our approach begins with a simple intuition: if two models $\mathcal{M}_1$ and $\mathcal{M}_2$ produce outputs that diverge substantially under the same input distribution, then their internal computations must also differ in many places. In particular, for any specific input, many activations in the trace of $\mathcal{M}_1$ cannot be reproduced correctly using the weights of *some other* model $\mathcal{M}_2$ whose output behavior diverges significantly from that of $\mathcal{M}_1$ (and $\mathcal{M}_2$ need not be known to the client). This observation leads to a simple

test for detecting whether the internal activations are inconsistent—and, in turn, whether the server’s claimed output is correct⁴.

Before describing the test, we recall that in a verifiable inference protocol, the client validates a claimed output using two inputs to the protocol: (i) a “ground truth” containing the weights and parameters of $\mathcal{M}_1$, and (ii) the messages sent by the server⁵. The second input may, of course, be adversarial (for clarity, we sometimes denote potentially malicious values with a tilde). For a simpler treatment, we first describe our solution in an *idealized* setting: we assume that at the beginning of the interaction the server gives the client oracle access to a long message, i.e., the client has random access to this message and can query only the parts it needs to remain efficient. The client has analogous random-access capability for its ground truth⁶. The concrete construction that keeps the Verifier efficient without this idealized model is described in Section 5.

A strawman and why it fails. Before describing our approach, it is instructive to first consider a simpler test. Suppose the client samples a random node a_j in the trace, reads the corresponding weights from the ground truth and the activation values of the nodes that “feed” into a_j , and checks that they are consistent: RandTestStrawman:

- Query the activation value a_j of a randomly chosen neuron.
- query the correct weights w_{ij} and activation values of the nodes a_i for $i \in G_j$
- Accept iff $a_j = \phi\left(\sum_{i \in G_j} w_{ij} a_i\right)$.

Our experiments, however, showed that discrepancies in the activations typically manifest only in “late” layers. This makes RandTestStrawman susceptible to false positives, as a random check in an early layer will likely pass even if the final output is different. Furthermore, while internal activations can be manipulated by an adversary, the input layer remains an immutable anchor.

The issues with RandTestStrawman suggest a different approach, which verifies the correctness of an entire *path* from input to output. By tracing a path from an output node to the input, we ensure the output is backed by a chain of consistent computation.

The construction. The protocol starts with the server sending oracles to the alleged activation values of the execution $(\tilde{a}_1, \tilde{a}_2, \dots)$. The client first samples a random node index j from the output layer (the rightmost layer in Figure 1). Then, it reads the following activation values from the trace sent by the server: $\tilde{a}_j$ together with all the activation values of j ’s parents (the nodes that feed into j); that is, the client also reads $(\tilde{a}_i)_{i \in G_j}$. From the ground truth string, the client will then read the weights $(w_{ij})_{i \in G_j}$, those relating node j to its parents in model $\mathcal{M}_1$. The client performs the following “local” check and rejects if this does *not* hold:

$$\tilde{a}_j = \phi\left(\sum_{i \in G_j} w_{ij} \tilde{a}_i\right) \quad (\star)$$

The client will then sample a random node index j' from G_j and proceeds as above (reading $\tilde{a}_{j'}$ and the nodes in $G_{j'}$, and performing the check in $(\star)$), until it reaches the input layer.

⁴Assume here for simplicity that the two models have the same architecture; we will see later that this assumption is not needed—see footnote 9.

⁵This is actually true in *any* Verifiable Computation protocol, where Prover and Verifier must agree on the “ground truth” of what function is being computed. Sometimes, like in our case, the description of the function is too large for the Verifier to handle and we provide the ground truth via some cryptographic tool, as we will see soon.

⁶This idealized presentation is standard in the cryptographic literature [Kil92, CHM⁺19, CHA24] and has the advantage of providing a modular treatment separating cryptographic concerns from information-theoretic ones.

At that point, if all the checks along the path passed, the client accepts the claimed value y as the valid output of $\mathcal{M}_1$ on qry .

If the server is providing a trace from some different model $\mathcal{M}_2$, we hypothesized that a randomly sampled path will fail this test with high probability. We validated our hypothesis by running extensive experiments which confirmed our intuition that this is indeed a good test: when comparing models with very different behaviors, we never experienced our test failing, therefore we believe that the probability of failure is negligible (see Sections 6 and 6.3 for details). This provides us with a lightweight but robust protocol template where it is sufficient to query and validate only a fraction of the network’s nodes. To cheat without being detected by this approach, the adversary would have to construct a trace with the correct input, the desired incorrect output, and internal activations such that many paths evaluate correctly according to the weights of the real model. We conjectured this to be hard. Indeed, our experimental analysis in Section 7 shows that two natural strategies (gradient descent and inverse transformations) fail to forge a trace that fools our test, even when the adversary is free to construct the trace arbitrarily (strong other-model soundness, Appendix H). A separate swap attack (Appendix F), which perturbs the true model’s output without involving any substitute model, tests a threat closer to full soundness and also fails.

Stronger security notions. The discussion so far focused on a relatively weaker security notion (which we dub *other-model soundness*) where we only consider cheating servers who run a different model $\mathcal{M}_2$. This seemingly restrictive notion is of practical and theoretical interest on its own. From an application standpoint, for example, whenever inference is offered as a cloud-based service, providers may have incentives in providing outputs from cheaper models, rather than the one advertised [D⁺25] (the ground truth). We refer the reader to Appendix A for more on this and other example scenarios. More broadly, other-model soundness captures settings where ad-hoc cheating strategies are practically infeasible or too expensive, making the use of an existing alternative model the most natural attack vector.

Other-model soundness has another appealing feature: as we show in Section 4.2, it lends itself very nicely to a *formal* treatment (in addition to the experimental treatment in Section 6). As we show in Theorem 1, two conditions are sufficient to guarantee security: (1) *trace separation*, meaning that two models with sufficiently dissimilar behavior have very different activation traces when run on the same input; (2) a “good” test that separates the two traces by sampling only a small number of locations of the trace and the ground truth. Whether these two conditions apply can, in principle, be assessed both theoretically and experimentally.

In addition to other-model soundness, we also consider a natural and stronger security notion of broader applicability, which we refer to as *full soundness*. Under this definition a server may act arbitrarily (may be “fully malicious”) and thus try more sophisticated strategies than running a different model. It could, in particular, provide an arbitrary output (not computed using any model) and commit to an arbitrary trace in C_{trc} . At least intuitively, full soundness is harder to achieve than other-model soundness. In fact, it escapes our formal treatment because it is not immediately clear, analytically, whether trace separation (which deals with traces from two different models) can have any implications for an arbitrary attack (Appendix A discusses this point as well).

We point out, however, that the analysis for other-model soundness trivially extends to full soundness for the case of “classification” models, i.e., models with a binary output (e.g., images of cats vs images of dogs) since the adversary can only “flip” the output. Between other-model soundness and full soundness lies *strong other-model soundness* (Appendix H), where the adversary runs a substitute model but may forge the trace arbitrarily. Our gradient descent and inverse transform attacks (Section 7) empirically test this intermediate

notion, while the swap attack (Appendix F) that perturbs the true model’s output without any substitute model is a threat closer to full soundness. Things get more complicated in the case of Large Language Models where the adversary can tinker with the output in more sophisticated ways, which make the detection probability of our “random path” test very low. We leave achieving full soundness for LLMs using our approach as an open problem. Formal definitions for the above security notions in the idealized model can be found in Section 4.1 and Appendix H; their cryptographic counterparts are in Section 4.3.

From idealized to concrete constructions. The above assumes for simplicity that the client has been given a direct random access to both the trace claimed by the server and the weights of the model. In our concrete scheme this is not the case; we implement this mechanism through cryptographic commitments. At the end of a training phase, a commitment $C_{\mathcal{M}}$ to the weights of the correct model ($\mathcal{M}_1$ above) is published in a public repository. This serves as the “ground truth” of what the correct model is. At the end of the protocol execution, the server publishes a commitment C_{trc} to the execution trace. At verification time the server provides the client with openings of the relevant values from the two commitments $C_{\mathcal{M}}$ and C_{trc} . Here we will use *vector* commitments [Mer87, CF13a], which guarantee that the server is bound to unique values and the opening is “succinct” (i.e., much smaller than the size of the network), yielding the desired efficiency for both client and prover. A general description of this compilation approach from idealized to cryptographic constructions, together with its security proof, can be found in Section 4.4.

Practical settings for approximate soundness. In many scenarios, even a relatively low soundness probability per query is acceptable. One reason is that inferences are produced at massive scale: if an adversary is routinely substituting models, the adversary will on average be caught after a moderate number of queries. Similarly, our approach is well-suited to auditing scenarios, where a verifier can issue many test queries against a model to which the provider has published a commitment.

Another reason is that providers may face severe penalties when caught cheating; a rational agent will not accept the risk even if per-query detection probability is low. This is especially natural in usage-based pricing settings (e.g., per-token billing for LLM APIs), where economic incentives to cut costs by serving a cheaper model are directly at play. More broadly, one could deploy our approach in settings where the user pays only if the result passes a verification check. This connects to the framework of *rational proofs* [AM12, CG15, CG17, CGG24], in which a prover is incentivized to behave honestly through a reward mechanism tied to verification rather than through cryptographic soundness alone.

3 Preliminaries

We now provide some preliminaries and notation that will be useful in describing our construction. Let (U, Y) be the input set and output set for a given inference, respectively. Consider a trained model $\mathcal{M}$ that runs on input queries $\text{qry} \in U$ and output $y = \mathcal{M}(\text{qry}) \in Y$. A client (which we will also call a *Verifier*) will ask a potentially untrusted server (a *Prover*) to run the model $\mathcal{M}$ on a particular query qry . Our goal is to prevent a malicious Prover (an *Adversary*) $\mathcal{A}$ to claim that $\hat{y} \neq y = \mathcal{M}(\text{qry})$ is the correct output of $\mathcal{M}(\text{qry})$. We assume that an adversary has access to the correct model $\mathcal{M}$ (its architecture and weights).

We will assume that models and queries are sampled from a distribution (see also Appendix C.2). When executed on a query qry , a model $\mathcal{M}$ can produce an evaluation trace trc (e.g., the list of all the activation values). We denote this mechanism through the syntax $\text{trc} := \text{EvalTrace}(\mathcal{M}, \text{qry})$. We assume that every trc is associated with a

unique output, intuitively the output $\mathcal{M}(\text{qry})$ of the model and query generating the trace. For this purpose we posit the existence of a function out with the property that $\text{out}(\text{EvalTrace}(\mathcal{M}, \text{qry})) = \mathcal{M}(\text{qry})$ ⁷. We require that both EvalTrace and out be efficiently computable. We stress that the output function always returns *some value* even if the trace is not fully well-formed or internally coherent: the output function is only trying to model the fact that every trace will implicitly be associated with some *claim* on the output of that trace. The output of the trace could for example be the output of the last layer of neurons, which is well-defined by the mere fact that this is an acceptable “trace”; clearly, this output is well-defined even if the prior layers are not in agreement with this output (in this case, somewhere in the trace there of course must be a contradiction). See also the formal treatment in the appendix (Appendix C and Remark 2).

If F is an algorithm and s is a string we use the standard convention F^s to denote the fact that F has oracle access (that is, random access) to the string s . Since F is not constrained to reading all of s , it can run in time *sublinear* in $|s|$.

We will occasionally talk about a “distance function” Δ and apply it in the same statement to both outputs and traces. In these situations we are implicitly assuming “a pair of distance functions”, respectively applied to outputs and traces. Talking about a single distance function allows us to use fewer subscripts without losing any essential points.

4 A Formal Framework for Verifiable Inference

In this section we present a formal framework for verifiable inference. Our approach borrows techniques from the *property testing* literature [Gol17], by leveraging the notion of similarity we discussed in Section 2. We assume that two models that produce very different outputs (they are *functionally dissimilar*) have “very different” traces (they are *representationally dissimilar*). We then show that this difference can be detected by a test that requires access only to a very small number of locations in the trace.

We first introduce our security definitions in an idealized model (Section 4.1), then provide a theoretical framework linking trace separation to other-model soundness (Section 4.2). We also present the cryptographic counterparts of these definitions (Section 4.3) and a generic compiler from idealized schemes to cryptographic ones using vector commitments (Section 4.4).

4.1 Idealized Verifiable Inference Schemes

We now present our framework for verifiable inference, which we will use to argue the security of our scheme.

Definition 1 (Inference Correctness Testing). Given a model $\mathcal{M}$ a *correctness tester* for $\mathcal{M}$ is an efficient algorithm TestTrace , with oracle access to some of its inputs, satisfying correctness and one of the two soundness definitions below (the former implies the latter).

Correctness. The scheme satisfies correctness if for any security parameter $\lambda \in \mathbb{N}$, model $\mathcal{M}$, query qry , we have:

$$\text{TestTrace}^{\mathcal{M}, \text{trc}}(1^\lambda, \text{qry}) = 1$$

where $\text{trc} := \text{EvalTrace}(\mathcal{M}, \text{qry})$. Correctness requires that the testing algorithm always accept the honestly generated trace.

⁷In natural cases out simply selects the appropriate neurons from the trace. See also Appendix C.

Full soundness. Let $\varepsilon : \mathbb{N} \rightarrow \mathbb{R}$ be a function and let $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}})$ be a pair of model and query distributions and Δ a notion of distance for probability distributions. We say that the scheme satisfies full (δ, ε) -soundness w.r.t to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ if for any adversary $\mathcal{A}$ and for any security parameter $\lambda \in \mathbb{N}$, we have:

$$\Pr \left[\begin{array}{l} \Delta[\mathcal{M}(\text{qry}), \text{out}(\widetilde{\text{trc}})] > \delta \wedge \\ \text{TestTrace}^{\mathcal{M}, \widetilde{\text{trc}}} (1^\lambda, \text{qry}) = 1 \end{array} : \begin{array}{l} \mathcal{M} \leftarrow_{\$} \mathcal{D}_{\text{model}} \\ \text{qry} \leftarrow_{\$} \mathcal{D}_{\text{query}} \\ \widetilde{\text{trc}} \leftarrow \mathcal{A}(1^\lambda, \mathcal{M}, \text{qry}) \end{array} \right] \leq \varepsilon(\lambda)$$

Informally, (full) soundness states that, except with ε probability, no adversary can fool the test into accepting when providing a trace with an output far from the correct one.

Other-model soundness. Let $\varepsilon : \mathbb{N} \rightarrow \mathbb{R}$ be a function and let $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ be as above and $\widetilde{\mathcal{D}}_{\text{model}}$ be a model distribution (potentially distinct from $\mathcal{D}_{\text{model}}$). We say that the scheme satisfies (δ, ε) -other-model-soundness w.r.t to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ and $\widetilde{\mathcal{D}}_{\text{model}}$ if for any security parameter $\lambda \in \mathbb{N}$, we have:

$$\Pr \left[\begin{array}{l} \Delta[\mathcal{M}(\text{qry}), \widetilde{\mathcal{M}}(\text{qry})] > \delta \wedge \\ \text{TestTrace}^{\mathcal{M}, \widetilde{\text{trc}}} (1^\lambda, \text{qry}) = 1 \end{array} : \begin{array}{l} \mathcal{M} \leftarrow_{\$} \mathcal{D}_{\text{model}} \\ \widetilde{\mathcal{M}} \leftarrow_{\$} \widetilde{\mathcal{D}}_{\text{model}} \\ \text{qry} \leftarrow_{\$} \mathcal{D}_{\text{query}} \\ \widetilde{\text{trc}} := \text{EvalTrace}(\widetilde{\mathcal{M}}, \text{qry}) \end{array} \right] \leq \varepsilon(\lambda)$$

Informally, other-model soundness states that, except with ε probability, the tester will be able to distinguish a false output whenever an adversary tries to use a different model (from a specific class) to evaluate qry .

In the next paragraphs, we will provide a theoretical framework to argue other-model soundness from first principles (based only on properties of the models involved and on the combinatorial properties of the testing mechanism). We will also later discuss how our experiments (and others from future work) are consistent with strong-other model soundness holding for the path-testing approach we propose in this work (as well as potentially other testing mechanisms with similar features).

4.2 Other-Model Soundness From Testing Trace Separation

We now formalize a general version of the assumptions related to similarity used in the AI literature. We will then formally show how these assumptions can allow us build an efficient **TestTrace** algorithm that detects if a different model is used to compute the wrong output.

To argue that our protocol has other-model soundness, we first formalize the *separability assumption* that underlies our approach: when sampling a model $\mathcal{M}$ and query qry from a certain distribution, any trace that significantly disagrees with the output $\mathcal{M}(\text{qry})$ will be “far enough” from the actual activation trace of $\mathcal{M}$ with high probability.

Definition 2 (Trace separation). Let $\mathcal{D}_{\text{model}}, \widetilde{\mathcal{D}}_{\text{model}}$ be model distributions and $\mathcal{D}_{\text{query}}$ be a query distribution. We say that they satisfy the $(\delta_{\text{sep}}^{\text{out}}, \delta_{\text{sep}}^{\text{trace}}, \varepsilon_{\text{sep}})$ -separation property with respect to a distance function Δ :

$$\Pr \left[\begin{array}{l} \Delta(\mathcal{M}(\text{qry}), \widetilde{\mathcal{M}}(\text{qry})) \geq \delta_{\text{sep}}^{\text{out}} \\ \implies \Delta(\text{trc}, \widetilde{\text{trc}}) \geq \delta_{\text{sep}}^{\text{trace}} \end{array} : \begin{array}{l} \mathcal{M} \leftarrow_{\$} \mathcal{D}_{\text{model}} \\ \widetilde{\mathcal{M}} \leftarrow_{\$} \widetilde{\mathcal{D}}_{\text{model}} \\ \text{qry} \leftarrow_{\$} \mathcal{D}_{\text{query}} \\ \text{trc} := \text{EvalTrace}(\mathcal{M}, \text{qry}) \\ \widetilde{\text{trc}} := \text{EvalTrace}(\widetilde{\mathcal{M}}, \text{qry}) \end{array} \right] \geq 1 - \varepsilon_{\text{sep}}$$

It is not enough that the traces follow distinct distributions; we must also have a good way to test it.

Definition 3 (Good separation test). Let $\varepsilon_{\text{tst}} : \mathbb{N} \rightarrow \mathbb{R}$ be a function. We say that a testing mechanism **Test** is ε_{tst} -good with respect to a distance function Δ and a threshold δ if for any parameter $\lambda \in \mathbb{N}$, model $\mathcal{M}$, query **qry**, $\widetilde{\text{trc}}$ such that:

$$\text{Test}^{\mathcal{M}, \text{trc}}(1^\lambda, \text{qry}) = 1 \quad (\dagger)$$

$$\Pr[\Delta(\text{trc}, \widetilde{\text{trc}}) \geq \delta \wedge \text{Test}^{\mathcal{M}, \widetilde{\text{trc}}}(1^\lambda, \text{qry}) = 1] \leq \varepsilon_{\text{tst}}(\lambda) \quad (\ddagger)$$

where $\text{trc} := \text{EvalTrace}(\mathcal{M}, \text{qry})$.

A good test will enable inference correctness testing as in Definition 1: condition $(\dagger)$ corresponds to correctness, while $(\ddagger)$ captures other-model soundness (with testing error $\varepsilon_{\text{tst}}(\cdot)$).

Theorem 1. Let $\mathcal{D}_{\text{model}}, \widetilde{\mathcal{D}}_{\text{model}}$ be model distributions and $\mathcal{D}_{\text{query}}$ be a query distribution. Then the following two conditions are sufficient (both should hold) for the existence of an inference correctness tester (Definition 1) with other-model-soundness:

- they satisfy the $(\delta_{\text{sep}}^{\text{out}}, \delta_{\text{sep}}^{\text{trace}}, \varepsilon_{\text{sep}})$ -separation property (Definition 2) with respect to a distance function Δ ;
- there exists a good testing mechanism with respect to Δ and threshold $\delta_{\text{sep}}^{\text{trace}}$ (Definition 3).

The resulting verifiable inference scheme has other-model soundness $\varepsilon(\lambda) \leq \varepsilon_{\text{tst}}(\lambda) + \varepsilon_{\text{sep}}$.

Proof (sketch). We can essentially employ the testing mechanism **Test** as a verifiable inference scheme. Correctness follows immediately from condition $(\dagger)$. Now, in the other-model soundness experiment in Definition 1, consider an adversary providing a model whose output is far from the honest one, that is such that $\Delta(\mathcal{M}(\text{qry}), \widetilde{\mathcal{M}}(\text{qry}))$. By invoking trace separation we know that the honest trace **trc** and $\widetilde{\text{trc}}$ will differ by at least $\delta_{\text{sep}}^{\text{trace}}$ (according to distance function Δ) with probability $(1 - \varepsilon_{\text{sep}})$. Invoking condition $(\ddagger)$, we can then conclude that the testing algorithm will reject with overall probability at least $(1 - \varepsilon_{\text{sep}}) \cdot (1 - \varepsilon_{\text{tst}}(\lambda))$. This provides the claimed soundness bound. $\square$

In Appendix I we describe how to empirically estimate $\delta_{\text{sep}}^{\text{out}}, \delta_{\text{sep}}^{\text{trace}}, \varepsilon_{\text{sep}}$, and ε_{tst} that can be used in Theorem 1.

4.3 Secure Verifiable Inference Protocol

We now define what a secure verifiable inference protocol in the cryptographic sense is. This notion can be seen as the “real-world” equivalent of our idealized notion in Definition 1 (we show how an idealized scheme can be generically compiled into cryptographic ones in Section 4.4). Note, firstly, that our definition relaxes the notion of soundness to accept outputs that may only be “approximately” correct, i.e., from an output distribution not too far from the one produced by the correct model. Also, for simplicity we present a definition for 3-round protocols, where the Prover sends the first and last messages, and the second round message is a random challenge (the structure of our instantiated protocol). It is not hard to extend this to an arbitrary interactive protocol between Prover and Verifier running on a commitment to the model $\mathcal{M}$.

Definition 4. A protocol for verifiable inference on a committed model is a tuple of algorithms (**GenParams**, **CommitToModel**, **Prove**, **Verify**) with the following syntax:

- $\text{GenParams}(1^\lambda) \rightarrow \text{pp}$: outputs security parameters;
- $\text{CommitToModel}(\text{pp}, \mathcal{M}) \rightarrow \text{cm}_{\mathcal{M}}$: outputs a commitment, or digest, to a model (deterministically).
- $\text{Prove}_1(\text{pp}, \mathcal{M}, \text{qry}) \rightarrow (\pi_1, \text{state})$: outputs a first proof message π_1 and a state for the next round, on input parameters, model and query.
- $\text{Prove}_2(\text{state}, \rho) \rightarrow \pi_2$: outputs the last proof message π_2 on input the state and a random challenge ρ (notice that without loss of generality the state can contain $\text{pp}, \mathcal{M}$ and qry).
- $\text{Verify}(\text{pp}, \text{cm}_{\mathcal{M}}, \text{qry}, y, (\pi_1, \pi_2), \rho)$: checks that the output of the model committed in $\text{cm}_{\mathcal{M}}$ on query qry is y , on additional inputs the security parameters, the challenge ρ and the proof messages π_1, π_2 .

These algorithms should satisfy binding, correctness and one of the soundness definitions described below. In these definitions we denote by $\mathcal{C}$ the space from which the random challenge is sampled.

Binding. For our notion to be meaningful, we require that a digest to a model is binding. Formally, for any efficient adversary $\mathcal{A}$, for every λ :

$$\Pr \left[\begin{array}{l} \mathcal{M} \neq \mathcal{M}' \wedge \text{cm} = \text{cm}' : \\ \text{pp} \leftarrow \text{GenParams}(1^\lambda) \\ (\mathcal{M}, \mathcal{M}') \leftarrow \mathcal{A}(\text{pp}) \\ \text{cm} \leftarrow \text{CommitToModel}(\text{pp}, \mathcal{M}) \\ \text{cm}' \leftarrow \text{CommitToModel}(\text{pp}, \mathcal{M}') \end{array} \right] \leq \text{negl}(\lambda)$$

Correctness. The scheme satisfies correctness if for any security parameter $\lambda \in \mathbb{N}$, model $\mathcal{M}$, query qry , we have:

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{GenParams}(1^\lambda) \\ \text{cm} \leftarrow \text{CommitToModel}(\text{pp}, \mathcal{M}) \\ \text{Verify}(\text{pp}, \text{cm}, \text{qry}, y, (\pi_1, \pi_2), \rho) = 1 : \\ (\pi_1, \text{state}) \leftarrow \text{Prove}_1(\text{pp}, \mathcal{M}, \text{qry}) \\ \rho \leftarrow_{\$} \mathcal{C} \\ \pi_2 \leftarrow \text{Prove}_2(\text{state}, \rho) \\ y := \mathcal{M}(\text{qry}) \end{array} \right] = 1$$

Full soundness. Let $\varepsilon : \mathbb{N} \rightarrow \mathbb{R}$ be a function and let $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}})$ be a pair of model and query distributions. Let Δ be a distance measure for distributions. We say that the scheme satisfies full (δ, ε) -soundness w.r.t to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ if for any efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and for any security parameter $\lambda \in \mathbb{N}$, we have:

$$\Pr \left[\begin{array}{l} \mathcal{M} \leftarrow_{\$} \mathcal{D}_{\text{model}} \\ \text{pp} \leftarrow \text{GenParams}(1^\lambda) \\ \text{cm} \leftarrow \text{CommitToModel}(\text{pp}, \mathcal{M}) \\ \Delta[\mathcal{M}(\text{qry}), \tilde{y}] > \delta \wedge \\ \text{Verify}(\text{pp}, \text{cm}, \text{qry}, \tilde{y}, (\pi_1, \pi_2), \rho) = 1 : \\ \text{qry} \leftarrow_{\$} \mathcal{D}_{\text{query}} \\ (\tilde{y}, \pi_1, \text{state}) \leftarrow \mathcal{A}(\text{pp}, \mathcal{M}, \text{qry}) \\ \rho \leftarrow_{\$} \mathcal{C} \\ \pi_2 \leftarrow \mathcal{A}_2(\text{state}, \rho) \end{array} \right] \leq \varepsilon(\lambda)$$

Other-model soundness. Let $\varepsilon : \mathbb{N} \rightarrow \mathbb{R}$ be a function and let $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ be as above and $\tilde{\mathcal{D}}_{\text{model}}$ be a model distribution (potentially distinct from $\mathcal{D}_{\text{model}}$). We say that the scheme satisfies ε -other-model-soundness w.r.t to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ and $\tilde{\mathcal{D}}_{\text{model}}$ if for any security parameter $\lambda \in \mathbb{N}$, we have:

$$\Pr \left[\begin{array}{l} \mathcal{M} \leftarrow_{\$} \mathcal{D}_{\text{model}} \\ \text{pp} \leftarrow \text{GenParams}(1^\lambda) \\ \text{cm} \leftarrow \text{CommitToModel}(\text{pp}, \mathcal{M}) \\ \Delta[\mathcal{M}(\text{qry}), \tilde{y}] > \delta \wedge \\ \text{Verify}(\text{pp}, \text{cm}, \text{qry}, \tilde{y}, : \text{qry} \leftarrow_{\$} \mathcal{D}_{\text{query}} \\ (\pi_1, \pi_2), \rho) = 1 \quad \tilde{\mathcal{M}} \leftarrow_{\$} \tilde{\mathcal{D}}_{\text{model}} \\ \tilde{y} := \tilde{\mathcal{M}}(\text{qry}) \\ (\pi_1, \text{state}) \leftarrow \text{Prove}_1(\text{pp}, \tilde{\mathcal{M}}, \text{qry}) \\ \rho \leftarrow_{\$} \mathcal{C} \\ \pi_2 \leftarrow \text{Prove}_2(\text{state}, \rho) \end{array} \right] \leq \varepsilon(\lambda)$$

Remark 1. In contrast to Definition 1, Definition 4 above has an explicit variable y referring to the claimed output by the prover. This is a mere cosmetic departure from Definition 1, however—in both notions the prover/adversary is claiming a well defined output. In the case of our idealized notion, this output corresponds to the value of $\text{out}(\text{trc})$. Adopting y kept the idealized setting slightly simpler and without redundant variables (the trace already contains output values; see also Remark 2). In Definition 4, we need an explicit y because we don't have fewer assumptions on the output of the adversary (we don't assume it is a trace anymore in the sense of Definition 5).

4.4 Compiling Idealized Schemes into Cryptographic Ones

Now we move away from the idealized model and show how to implement our protocol in practice, by constructing a secure way to give the Verifier random access to the parameters of the correct model, and to the trace of the computation of the adversary. In particular, we present a compiler that transforms an idealized scheme (in the sense of Definition 1) into its cryptographic counterpart (Definition 4) through general vector commitments [CF13b]. This approach is standard in the cryptographic literature.

A *vector commitment (VC)* is a cryptographic primitive that allows a prover to commit to a tuple $v = (v_1, v_2, \dots, v_n)$ in such a way that the resulting commitment is *binding* (the prover cannot later open it to a different sequence) and *positionally binding* (each committed value is bound to its index in the vector). Later, the prover can efficiently produce a short proof—called an *opening*—that reveals the value at a specific position i and convinces a verifier that this value is indeed the one that was committed at that position, without revealing the other entries in the vector.⁸ A popular and foundational construction for vector commitments is the Merkle tree [Mer89], which we adopt in our scheme (in an optimized variant).

Let Π_{VecCom} be a vector commitment scheme and Π_{ideal} be an idealized verifiable inference scheme. Let $(n_{\text{model}}, n_{\text{qry}}, n_{\text{trc}}, \text{EvalTrace}, \text{Idxs}_{\text{out}})$ be a family of models (as defined in Definition 5). The final scheme would work as follows:

- *Parameter generation:* runs the parameter generation algorithm of Π_{VecCom} (if the algorithm requires a length parameter, use the maximum expected length of any trace).

⁸We provide a more formal definition in Appendix B.

- *Model commitments:* Run the commitment algorithm of Π_{VecCom} on a string representation of $\mathcal{M}$.
- *Prover:*
 1. Generate $\text{trc} := \text{EvalTrace}(\mathcal{M}, \text{qry})$ and use Π_{VecCom} to commit to it. Let cm_{trc} be the resulting commitment.
 2. Send $\pi_1 := \text{cm}_{\text{trc}}$; receive random challenge ρ .
 3. Use string ρ as randomness for running the idealized testing algorithm

$$\Pi_{\text{ideal}}.\text{TestTrace}^{\mathcal{M}, \text{trc}}(1^\lambda, \text{qry});$$

4. Let $\mathcal{S}_{\mathcal{M}}^{(\text{pos})}$ and $\mathcal{S}_{\text{trc}}^{(\text{pos})}$ be the sets of indices that the execution TestTrace queries for the string $\mathcal{M}$ and trc respectively. Let $\tilde{\mathcal{S}}_{\text{trc}}^{(\text{pos})} := \mathcal{S}_{\text{trc}}^{(\text{pos})} \cup \text{Idxs}_{\text{out}}$. Use the opening algorithm of Π_{VecCom} to open each of the values in $(\mathcal{M}[j])_{j \in \mathcal{S}_{\mathcal{M}}^{(\text{pos})}}$ (resp. in $(\text{trc}[j])_{j \in \tilde{\mathcal{S}}_{\text{trc}}^{(\text{pos})}}$) against commitment $\text{cm}_{\mathcal{M}}$ (resp. cm_{trc}). Let openings denote the concatenation of all these opening proofs as well as the related values from $\mathcal{M}$ and trc .
 5. Return $\pi_2 := \text{openings}$
- *Verifier:*
 1. Use string ρ as randomness for running the idealized testing algorithm

$$\Pi_{\text{ideal}}.\text{TestTrace}^{\mathcal{M}, \text{trc}}(1^\lambda, \text{qry})$$

in the following way: whenever there is an access to $\mathcal{M}$ or trc , use the corresponding value in openings for it; if that value has not been provided by the prover reject. If TestTrace does not reject, go to the next step; otherwise, reject.

2. Check all the opening proofs in openings and accept if and only if they are all valid and y is consistent with the trace output found in openings .

The following theorem links the security of idealized verifiable inference schemes (Definition 1) and of vector commitments (Appendix B) to that of cryptographic verifiable inference schemes (Definition 4).

Theorem 2. *Let $\mathcal{D}_{\text{model}}, \tilde{\mathcal{D}}_{\text{model}}$ be model distributions and $\mathcal{D}_{\text{query}}$ be a query distribution. Let Π_{VecCom} be a vector commitment scheme and Π_{ideal} be an idealized verifiable inference scheme, then:*

- *if Π_{ideal} is fully ε -sound with respect to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ then the construction above is a verifiable inference scheme that is fully $(\varepsilon(\cdot) + \text{negl}(\cdot))$ -sound with respect to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$.*
- *if Π_{ideal} satisfies ε -other-model-soundness with respect to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ and $\tilde{\mathcal{D}}_{\text{model}}$, then the construction above is a verifiable inference scheme that is $(\varepsilon(\cdot) + \text{negl}(\cdot))$ -other-model-sound with respect to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ and $\tilde{\mathcal{D}}_{\text{model}}$.*
- *if Π_{ideal} satisfies ε -strong-other-model-soundness with respect to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ and $\tilde{\mathcal{D}}_{\text{model}}$, then the construction above is a verifiable inference scheme that is $(\varepsilon(\cdot) + \text{negl}(\cdot))$ -strong-other-model-sound with respect to $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ and $\tilde{\mathcal{D}}_{\text{model}}$.*

Proof (sketch). This compilation approach is standard and we do not propose a full proof here. See [Kil92] or [BSCS16]. The completeness and soundness of the protocol above follow immediately. Binding follows directly from that of the vector commitment. $\square$

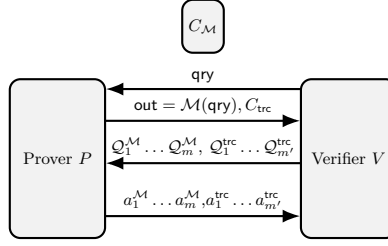


Figure 2: Interaction flow in our protocol. The Prover computes $\text{trc} = \text{EvalTrace}(\mathcal{M}, \text{qry})$ and commits to it via a vector commitment $C_{\text{trc}} = \text{VC}(\text{trc})$. Verifier runs **TestTrace** where the queries of the test algorithm to $\mathcal{M}$, trc are answered by the Prover through the vector commitments $C_{\mathcal{M}}, C_{\text{trc}}$. See also the alternative protocol description in Section 5.

5 Our Main Construction

In this section we present our construction and discuss its security.

5.1 The Construction

Before describing our construction, we briefly recall the relevant structure of neural networks (introduced in Section 2) and some other items useful to understand how the construction works.

- A neural network is organized into d layers, numbered 1 (input) through d (output). Each neuron j in layer ℓ receives inputs from a set G_j of parent neurons in layer $\ell - 1$, and its activation is computed as $a_j = \phi(\sum_{i \in G_j} w_{ij} a_i)$, where w_{ij} are the edge weights and ϕ is a non-linear activation function.
- We consider the *execution trace* trc of an inference as the vector of all activation values $(a_1, a_2, \dots)$, one per neuron, organized so that each entry is associated with a specific layer and has a well-defined set of parents.
- The *architecture* of the network—i.e., the graph structure specifying which neurons connect to which, the number of layers, and the layer each neuron belongs to—is public and known to both the Prover and the Verifier. The committed values are the *weights* $(w_{ij})_{ij}$ and the *activations* $(a_1, a_2, \dots)$; the wiring itself is not secret. This means that when either party refers to a parent set G_j or samples a random neuron in a given layer, both sides agree on what that means.

In Figure 3 we describe **RandPathTest**, the idealized version of our protocol. As sketched in Section 2, the test samples a random path from an output node back to the input layer and checks, at each node along the path, that the claimed activation is locally consistent with the correct model’s weights and the claimed activations of the node’s parents.

In Figure 4 we describe our final cryptographic protocol (the reader can also find a high-level perspective of the protocol flow in Figure 2). As shown in Figure 4, access to the correct weights is provided via a VC $C_{\mathcal{M}}$ which we assume has been computed correctly. However, note, the commitment to the trace C_{trc} is generated by the adversary, so we have no guarantee that it is indeed the actual trace of the computation of the expected model. This construction can be seen as an application of a specific instance of the generic compiler using vector commitments described in Section 4.4⁹.

⁹ This test also enforces that whatever the Prover committed to in C_{trc} has to be a trace on a model with the same architecture as $\mathcal{M}$, therefore we do not need to assume that is the case—see also Footnote 4.

$\text{RandPathTest}^{\mathcal{M}, \widetilde{\text{trc}}}(1^\lambda, \text{qry})$:

Input: Oracle access to model weights $\mathcal{M}$ and claimed trace $\widetilde{\text{trc}}$; query qry .

- Sample a node $\tilde{a}_{j_d} \in \text{out}(\widetilde{\text{trc}})$ where d is its depth.
- For layers $\ell = d - 1, \dots, 1$:
 - Query $\tilde{a}_i \in \widetilde{\text{trc}}$ for all $i \in G_{j_\ell}$ (the claimed activation values of the nodes that feed into node j_ℓ) and the correct weights w_{ij_ℓ} from $\mathcal{M}$ for $i \in G_{j_\ell}$. Note that all activations in G_{j_ℓ} are in layer $\ell - 1$.
 - Sample $j_{\ell-1}$, a random index in G_{j_ℓ} , and repeat.
- Accept iff for all $\ell = d - 1, \dots, 1$: $\tilde{a}_{j_\ell} = \phi\left(\sum_{i \in G_{j_\ell}} w_{ij_\ell} \tilde{a}_i\right)$.

Figure 3: The **RandPathTest** algorithm (our protocol in the idealized model). The test samples a random path from an output node back to the input layer and checks local consistency at each node along the path using the correct model’s weights.

5.2 Security

We now discuss security properties of our construction. Since by Theorem 2, the compiled cryptographic construction inherits the guarantees of the idealized protocol, in the following discussion we simply focus on the idealized **RandPathTest**.

Other-model soundness. Recall that under other-model soundness, the adversary is restricted to running a different model $\tilde{\mathcal{M}}$ and committing to its honest execution trace. Intuitively, if $\mathcal{M}$ and $\tilde{\mathcal{M}}$ produce sufficiently different outputs, their activation traces must also diverge at many internal nodes, making the discrepancy detectable by sampling a random path.

To establish this formally, one may apply our theoretical framework from Section 4. By Theorem 1, two conditions suffice: (1) trace separation (Definition 2), meaning that models with dissimilar outputs produce traces that are far apart; and (2) the existence of a good separation test (Definition 3), meaning that this distance is detectable from a small number of sampled positions. If both conditions hold, the resulting scheme achieves other-model soundness with error $\varepsilon(\lambda) \leq \varepsilon_{\text{tst}}(\lambda) + \varepsilon_{\text{sep}}$.

We validate both conditions empirically. Our model separation experiments (Section 6) confirm that common architectures—ResNet-18 classifiers and Llama-2-7B—exhibit trace separation: models with different training or fine-tuning produce measurably different activation patterns, with separation values bounded well away from zero across all tested paths. The same experiments confirm that **RandPathTest** successfully distinguishes these traces, acting as a good separation test. We therefore conclude that our protocol satisfies other-model soundness for the model families and input distributions tested.

Strong other-model soundness. Our adversarial experiments in Section 7 provide evidence that **RandPathTest** resists natural attack strategies that go beyond the other-model soundness threat model. Under strong other-model soundness (defined formally in Appendix H), the adversary still uses a substitute model $\tilde{\mathcal{M}}$ to produce the output but is free to construct the committed trace arbitrarily, rather than submitting the honest trace of $\tilde{\mathcal{M}}$. This captures a natural threat scenario: an adversary who serves a cheaper or dealigned model and then attempts to forge a trace that passes verification against the committed weights of $\mathcal{M}$.

The Final Protocol:

- **Setup (commitment to the model).** A digest of the model $\mathcal{M}$ is provided to the client via a vector commitment $C_{\mathcal{M}}$ to the weights $(w_{ij})_{ij}$ of the neural network (where w_{ij} is the weight on the edge from neuron i to neuron j). We assume $C_{\mathcal{M}}$ to be correctly computed at the end of the training phase. The Verifier holds $C_{\mathcal{M}}$; the Prover holds the decommitting information for $C_{\mathcal{M}}$.
- **Proving stage.** On input query qry :
 1. The Prover computes $\text{trc} := \text{EvalTrace}(\mathcal{M}, \text{qry})$, the execution trace consisting of all activation values $(a_1, a_2, \dots)$ produced during inference. It sends the Verifier the claimed output $\text{out}(\text{trc})$ together with a vector commitment C_{trc} to the entire trace.
 2. The Verifier sends a random challenge ρ .
 3. Using ρ as randomness, the Prover determines a random path from an output node back to the input layer as done in Figure 3. This identifies a set of positions in $\mathcal{M}$ (weights along the path) and in trc (activations along the path and their parents). During this stage, the Prover opens the corresponding entries from $C_{\mathcal{M}}$ and C_{trc} and sends the values and opening proofs to the Verifier.
- **Verification stage.** The Verifier:
 1. Checks that all vector commitment openings are valid with respect to $C_{\mathcal{M}}$ and C_{trc} ; rejects if any opening fails.
 2. Using randomness ρ , reconstructs the same path as the Prover. For each node j_ℓ on the path, verifies the *local consistency check*: that the claimed activation $\tilde{a}_{j_\ell}$ equals $\phi\left(\sum_{i \in G_{j_\ell}} w_{ij_\ell} \tilde{a}_i\right)$, where the weights w_{ij_ℓ} come from $C_{\mathcal{M}}$ and the parent activations $\tilde{a}_i$ come from C_{trc} .
 3. Accepts if and only if all local consistency checks pass.

Figure 4: Our concrete cryptographic protocol for verifiable inference. The protocol instantiates the generic compiler of Section 4.4 with **RandPathTest** (Figure 3) as the testing mechanism. We instantiate vector commitments with a Merkle-tree-based approach.

We evaluated three distinct attack strategies—gradient-descent reconstruction, inverse transforms, and logit swapping—each designed to produce a trace that is consistent with the correct model’s weights yet yields an incorrect output. Across millions of evaluated paths, none succeeded: all produced traces with separation values bounded well away from zero at every layer. The fundamental obstacle is that fooling **RandPathTest** requires constructing a trace that is simultaneously consistent along many paths from output to input, a constraint that tightens combinatorially with network depth and width. We stress that these results demonstrate resistance to specific attack strategies in the context of specific tasks, not a guarantee against all possible adversaries in any setting; stronger attacks may exist.

The swap attack (Appendix F) tests a threat that is closer to full soundness: the adversary takes the true model’s output, perturbs it by swapping the highest- and lowest-probability tokens, and then optimizes intermediate activations to match. Even in this setting, 100% of attack attempts produced measurable activation deviations (minimum separation value of 0.031), indicating that **RandPathTest** detects tampering even when no

explicit substitute model is involved.

Full soundness. Achieving full soundness—where the adversary may produce an arbitrary output and construct the trace in any way it chooses—remains in part an open problem. A fundamental limitation of any single-path sampling strategy is that detection probability cannot exceed $1/N$, where N is the maximum layer width of the network. To see why, observe that an adversary can modify a single node in a layer of size N , and a single random path will select that node with probability exactly $1/N$. While this bound does not appear to be a practical limitation under (strong) other-model soundness—where our experiments show detection across all tested paths—it may become relevant when the adversary is free to choose arbitrary outputs and concentrate modifications in as few nodes as possible.

A natural mitigation is to sample multiple independent paths, which increases the detection probability at the cost of larger proofs and more communication. One can also consider adaptive sampling strategies that prioritize layers or nodes where activations are statistically more sensitive to tampering (we have some partial results from this approach and we are currently developing them further). Understanding the precise tradeoffs between the number of sampled paths, proof size, and the resulting soundness guarantees is an important direction for future work.

6 Model Separation Experiments

To show that our protocol satisfies other-model soundness (Section 4.2), we must show that the underlying assumptions of separability hold. In this section, we detail our efforts to empirically evaluate this assumption through a series of experiments on common machine learning models.

6.1 Experiment Design

To validate our assumptions, our experiments must answer two questions:

1. Can models with identical architectures but different training data, or models trained on the same data where one is fine-tuned, be distinguished based on their evaluation trace (i.e., the list of all activation values)?
2. Is our path testing protocol **RandPathTest** (Section 5) able to distinguish between two models with different evaluation traces?

The first question relates to the notion of trace separation, whereas the second question helps us understand if our protocol is a good separation test.

In our experiments, to calculate the differences between model traces, we computed the activation differences at each layer. For a verified model $\mathcal{M}$ and an unverified model $\widetilde{\mathcal{M}}$, and for all $1 \leq i \leq L$, this difference, which we call the **separation value**, is computed as:

$$|\phi((a_{\mathcal{M}}^{i-1})^\top W_{\mathcal{M}}^i) - \phi((a_{\widetilde{\mathcal{M}}}^{i-1})^\top W_{\mathcal{M}}^i)| \quad (1)$$

To answer the first question, we select a model $\mathcal{M}$ and a similar model $\widetilde{\mathcal{M}}$, and compute Equation (1) for all layers in the network. If we can see a marked difference between the two models then we can claim trace separation.

To answer the second question, instead of looking at the entire model, we focus on a single, specific path through the network. So, we replace the activations of $\mathcal{M}$ along a random path P with those from $\widetilde{\mathcal{M}}$, and then recompute Equation (1).

In the remainder of this section, we discuss our model separation experiments on two specific architectures: a pre-trained ResNet-18 classifier model and the Llama-2-7B LLM.

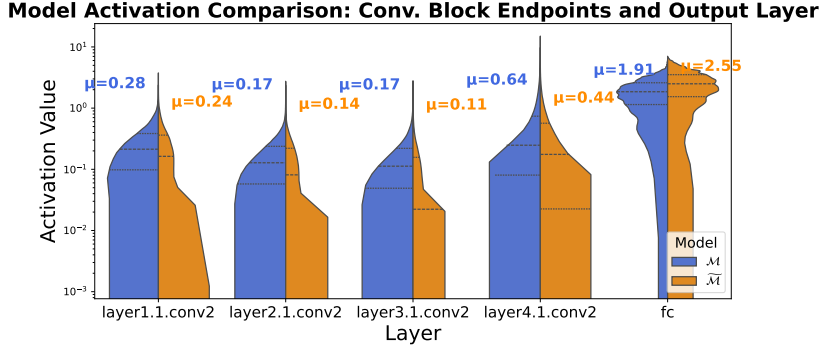


Figure 5: Activation distributions for path-selected neurons at bottleneck block endpoints and FC layer across models. Split violin plots show the probability density for neurons traversed during path selection. $\mathcal{M}$ (blue) and $\tilde{\mathcal{M}}$ (orange) distributions are separated with quartile markers and annotated mean values.

6.2 Model Separation in Classifiers

6.2.1 Setup

We selected ResNet-18, a widely-used architecture for classification problems [HZRS16]. When designing the experiments we wanted to use a lightweight yet popular model to ensure that our results are interpretable, transparent and general. We created two instances of this model: $\mathcal{M}$, which classifies images of dogs vs cats, and $\tilde{\mathcal{M}}$ for dogs vs squirrels.¹⁰

Since we seek to measure the trace separation between the two models, the shared category dog provides a way of linking the performance of the two models. Both $\mathcal{M}$ and $\tilde{\mathcal{M}}$ achieved similar performance on the common task; they achieved 95% and 98% accuracy, respectively, on this task, with a 95.3 and 98.4% F1-score. The presence of the dog class allowed us to assess how internal representations vary between models with the same architecture but contrasting training classes. We hypothesized that this training difference would show up as a detectable and measurable divergence in their internal activation patterns, even when both were processing images from the common dog class.

To answer the questions in Section 6.1, we evaluate, using Equation (1), activation differences (1) across paths through the models and (2) across both models overall. Given the relatively small size of the model (ResNet-18 contains approximately 11M parameters), analyzing all activation values in addition to individual paths is feasible. In our analysis, we focus on two classes of layers: those for *feature extraction* (i.e., Conv2d layers at various depths) and those for *classification* (the final layers including the linear layer that produces the binary classification output).

To ensure robust statistical analysis, we used activations from 3,522 test images for dogs. The images used were those that were positive instances in both models.

6.2.2 Results

We first analyzed the distribution of activation differences that occurred during inference on dog images¹¹ when the activations along the random path from the unverified model ($\tilde{\mathcal{M}}$) were substituted into the verified model ($\mathcal{M}$).

¹⁰Training data: Dog, cat, and squirrel classes extracted from Animals-10 dataset from <https://www.kaggle.com/datasets/alessiocorrado99/animals10>

¹¹From <https://www.kaggle.com/datasets/chetankv/dogs-cats-images>.

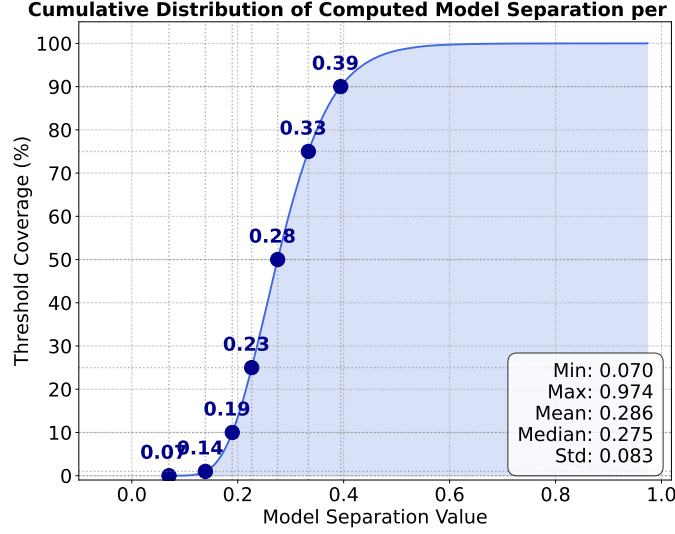


Figure 6: Cumulative distribution of model separation values (Equation (1)) for path-selected neurons, aggregated by mean per image-round path. The x-axis shows separation magnitudes; the y-axis shows the percentage of paths below each threshold. Blue dots indicate percentile thresholds.

Activation differences for paths. As demonstrated by the violin plot in Figure 5, the initial feature-extraction layers show almost no difference in activations because both models learned comparable low-level features for dog recognition. However, the distributions greatly expand as data moves to deeper, more abstract layers. Despite reaching the same result, the models still diverge. This is particularly noticeable in the last few layers of the classifier, lending support to the trace separation property.

We examined the cumulative distribution of mean model separation values across 176,100 test instances ($3,522 \times 50$ random path tests for each), which is presented in Figure 6. The minimum separation value, computed using Equation (1), is 0.070. Although the two models had an similar classification score on the same task, this statistical analysis revealed measurable internal differences in their input processing.

This pattern is also confirmed by the histogram in Figure 7 of all activation values across 176,100 paths. The pattern is subtle at the initial layers, but increasingly shows much different distributions between the activation in the deeper layers. This reveals that simply looking at the output can mask many characteristic differences in their traces.

Activation differences across layers. We processed the activation values of path-selected neurons layer-by-layer using Jensen-Shannon divergence (JS) ¹². The JS between the activation distributions of both models for path-selected neurons was low in the initial convolutional layers (conv1: 0.0, layer1.1.conv2: 0.198), indicating moderate representational overlap. The divergence values increased through the middle layers, reaching maximum at layer3.1conv2 (0.285), then decreased in the deeper layers (layer4.1.conv2: 0.093, fc: 0.245). The peak divergence in the middle layers corresponds to differences in intermediate feature representations, while the final classifier layer (fc: 0.245) shows how each model created decision boundaries to separate classes. These results show that the two models do exhibit measurable differences at certain layers, which means a path

¹²Jensen-Shannon divergence (JSD) measures the similarity between two probability distributions. It is a symmetric and smooth version of Kullback-Leibler divergence (D_{KL}), defined as $JSD(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(Q \parallel M)$, where $M = \frac{1}{2}(P + Q)$.

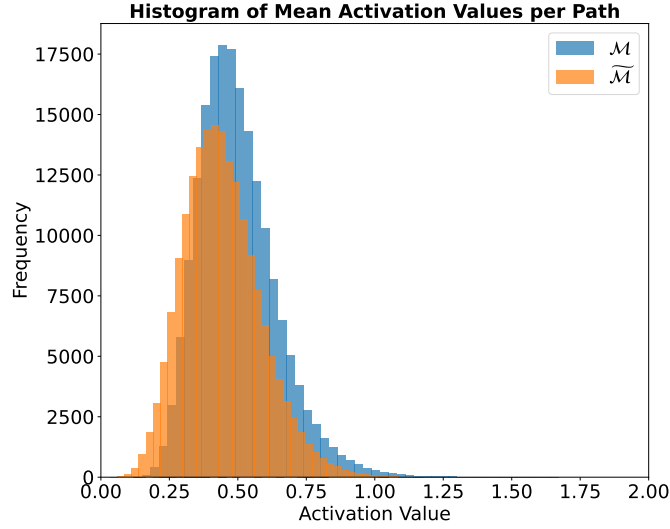


Figure 7: Mean activation values per path. Each point is the average activation across layers for one image-round pair.

through these layers will also detect.

Additional experiments We discuss additional separation results on classifiers in Appendix G.

6.3 Model Separation in LLMs

6.3.1 Setup

To answer the questions in Section 6.1 in the setting of LLMs, we study Llama-2-7b-chat-hf ($\mathcal{M}$) and Llama-2-7b-hf ($\tilde{\mathcal{M}}$), two widely-used open-source models for general text generation [ea23]. Because the research community has used them extensively, our results on them are more readily interpretable. Both contain 7 billion weights across 32 transformer layers with hidden dimension of 4096, processed using FP16 precision on the GPU.

As in the ResNet-18 experiments above, we wish to establish that $\mathcal{M}$ and $\tilde{\mathcal{M}}$ are similar. Our ResNet-18 experiments are oriented around demonstrating model separability clearly by comparing binary classifiers with a common class. But, with our LLM experiments, we wish to show a scenario closer to a real-world deployment for verifiable AI: separability between a fine-tuned model and its base version.

We extracted each LLM’s weights and measured the weight differences using the Frobenius norm. The models were closely aligned: cross-layer parameter analysis revealed high consistency across the model’s 291 weight matrices, with a mean relative Frobenius norm of 0.052.¹³

In this experiment, we analyze model separation between across selected paths. To validate the formula (Equation (1)) the experimental setup utilized forward hooks to capture intermediate activations during inference, allowing us to analyze internal representations. Unlike our ResNet-18 experiments, we do not also analyze the activation differences across both LLMs as these models are prohibitively large to study. We argue that showing

¹³Normalization layers showed the highest consistency (0.010 ± 0.003). MLP and attention layers had similar magnitudes (0.063–0.066) with attention layers exhibiting greater variance (± 0.014 vs ± 0.003), and the output layer showing expected higher deviation (0.124) due to its specialized function.

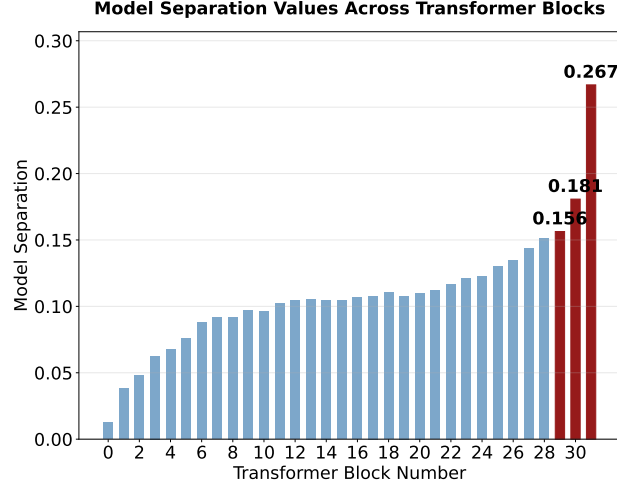


Figure 8: Model separation values across transformer blocks between Llama-2-7b-hf and Llama-2-7b-hf-chat models. The chart shows the mean separation value (Eq. 1) for each transformer block. Top 3 blocks are in red. Dataset: 25 sequences.

divergence in the path activations is sufficient to establish trace separation; indeed, if we can show that one path through the activation traces is different, then necessarily the traces are different.

In each trial, we select neurons at each layer to form path through the network and measure the activation differences between the two models. We also verified that selecting the neuron with minimum separation value (most similar) at each layer yields consistent results, confirming that the observed separation is robust across different path selection strategies.

6.3.2 Result

We recorded intermediate representations at each layer during inference on 25 token sequences. This led to 290 layer-wise comparisons per test case. The separation values showed consistent patterns that diverged markedly from the weight similarity, owing to nonlinear activation functions.

The model separation values revealed a clear progressive pattern across the model’s 32 transformer blocks (Figure 8). Early blocks (0–8) showed minimal separation with mean values around 0.013–0.092, suggesting that basic feature extraction remains largely unchanged. Middle layers (9–24) exhibited a gradual increase in separation values, reaching around 0.096–0.123. Late layers (25–31) demonstrated sharp spikes in separation, particularly in blocks 25 (0.130), 30 (0.181), and 31 (0.267). The final block (31) exhibited the highest mean separation value of approximately 0.267, indicating that the final representations between the models are the most different. These results show that as earlier blocks capture low-level features, it is harder to distinguish between models, while later blocks process more complex features and show higher separation values, making it easier to distinguish between models that differ in training. This supports the notion of trace separation. Equation (1) effectively differentiates between closely related models (Figure 9). The model separation analysis shows that the smallest separation is approximately 0.013 (Block 0) and the median separation is approximately 0.105. All of the separation values are greater than 0.01, which means that the differences are clearly distinguishable, hence the experiment design is sound. This shows that our method can reliably find variations in activation patterns across transformer layers, even in models that are very similar in

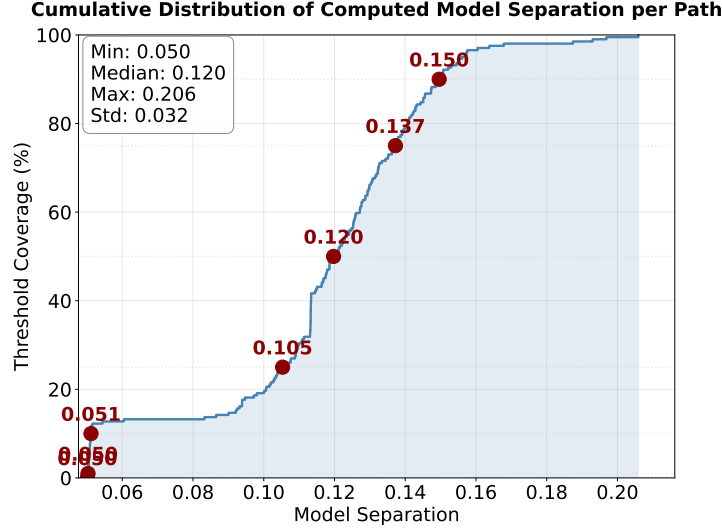


Figure 9: Cumulative distribution of model separation (Equation (1)) between Llama-2-7b-hf and Llama-2-7b-hf-chat, focusing on the last token (output) for each sequence (x: separation value, y: ratio of tests below that threshold, red dots: percentile thresholds). Neurons are randomly selected per layer, with mean values aggregated per path.

structure. These results demonstrate that our path testing protocol was able to distinguish between two models with different evaluation traces, confirming that the observed separation is robust across different path selection strategies.

7 Adversarial Attacks on Model Separation

We now experimentally investigate easy and efficient attacks which could be mounted against the protocol. To do this, an adversary using $\tilde{\mathcal{M}}$ would need to emulate the valid model in such a way that the commitments to its activations which give the result y_A would seem to be those produced from $\mathcal{M}$.

7.1 Attack Design

We designed attacks to determine how difficult it is for an adversary to forge a trace that passes `RandPathTest` against the committed model $\mathcal{M}$. To evaluate the efficacy of each attack, we computed the model separation values at each layer using Equation (1). For the evaluation criteria during path testing, the mean of the set of separation values $\{s_i : 1 \leq i \leq L\}$ should be zero, where s_i denotes the separation value at layer i and L is the total number of layers. High separation values indicate that the adversary failed to reconstruct the verified model’s activations.

While the model separation experiments of Section 6 show that submitting the honest trace of a substitute model $\tilde{\mathcal{M}}$ is detected (other-model soundness), a more capable adversary might try to *forge* a trace: run $\tilde{\mathcal{M}}$ to obtain the output, then construct an arbitrary trace that passes `RandPathTest` against $\mathcal{M}$ ’s weights. This corresponds to the strong other-model soundness notion defined in Appendix H.

This threat model is practically relevant: an adversary who substitutes a cheaper, quantized, or dealigned model knows $\mathcal{M}$ ’s committed weights and can attempt to reverse-engineer activations consistent with them. We explore two natural reconstruction strategies:

7.1.1 Gradient Descent

Gradient descent is used during training to adjust the weights of a model through an iterative process. As such, we wish to determine if an adversary can reconstruct the activations in this way, using gradient descent to optimize over *activations* rather than weights. To do so, we first compute: $\text{out} \leftarrow \mathcal{M}(\text{qry})$, then calculate the loss as $L = (\mathcal{M}(\text{qry}') - \text{out})^2$ and $\text{qry}'_i = \text{qry}'_{i-1} - \alpha \frac{\partial L}{\partial \text{qry}'_{i-1}}$, with α being a learning rate. We repeat this process until the gradient descent converges.

Our test in Section 7.2 looks at the possibility that the adversary can perform a reconstruction attack by gradient descent on two different models: a simple ANN and the Llama2-7B LLM. We hypothesized that simpler models with fewer weights might exhibit a higher susceptibility to adversarial reconstruction, as their reduced complexity could help an attacker’s ability to reverse-engineer inputs from observed activations, given knowledge of the data distribution.

A special case of this attack involves an adversary who attempts to change the semantics of LLM output by swapping output tokens and then trying to reconstruct activations to falsify a proof that the output came from the original model. Due to space constraints, we discuss this attack in Appendix F.

7.1.2 Inverse Transform

We also explored the feasibility of an inverse transform attack using a simple ANN architecture. Due to space constraints, we discuss this attack in Appendix E.

7.2 Gradient Descent Attack on Basic Models

In this experiment, we use a simple, custom model to investigate whether it is possible to reconstruct internal activation structure of a neural network using only model’s weights and observed output via gradient descent.

7.2.1 Setup

The custom model we used in this experiment as our target was intentionally designed to make the adversarial attack as simple as possible: a 3-layer fully connected ANN using min-max scaling (so all values are in the range $[0, 1]$), trained on a dataset containing 150 samples and 4 features for 3 species of irises¹⁴. The input layer takes in four features, the two hidden layers have 64 and 32 neurons each and use ReLU activation functions, and the output layer has 3 neurons for classification.

The adversarial attack was configured to run 10,000 epochs with a learning rate of 0.005, using mean squared error (MSE) loss to minimize the difference between target and predicted activations. We conducted experiments with 125 different input samples, each tested across 50 rounds to ensure statistical validity, as the attack is non-deterministic.

7.2.2 Results

Our results can be found in Table 1, where we report the minimum, mean, and maximum separation values across each of the three layers in our custom model, calculated using Equation (1). The table contains pass rates, i.e., the percentage of adversarially constructed activations whose separation values fall under each threshold. Since the adversary’s goal is to achieve zero separation, high pass rates at low thresholds indicate weak soundness (adversary succeeds), while low pass rates indicate strong soundness (verification system successfully detects adversarial inputs). These thresholds thus represent a tradeoff between

¹⁴https://scikit-learn.org/stable/datasets/toy_dataset.html#iris-dataset

Table 1: % of Samples Passing Threshold by Metric & Layer

Metric	Layers	Threshold			
		10^{-6}	6.7×10^{-5}	4.6×10^{-3}	0.308
Minimum	L_1	100.0%	100.0%	100.0%	100.0%
	L_2	100.0%	100.0%	100.0%	100.0%
	L_3	0.1%	5.1%	99.9%	100.0%
Mean	L_1	0.0%	0.0%	2.5%	100.0%
	L_2	0.0%	0.0%	27.5%	100.0%
	L_3	0.0%	0.0%	98.7%	100.0%
Maximum	L_1	0.0%	0.0%	0.0%	90.0%
	L_2	0.0%	0.0%	0.4%	100.0%
	L_3	0.0%	0.0%	96.7%	100.0%

completeness and soundness: the lower the threshold, the stricter the verification criteria and the better the soundness, though overly strict thresholds may reject legitimate inputs and impact completeness due to floating point error.

Based on the results, there is a steep drop in pass rates as the reconstruction target moves from hidden layers toward the output layer. When examining minimum separation values – the best-case scenario for the adversary – the earlier layers show that adversarial reconstruction succeeds: L_1 and L_2 both achieve 100% pass rates at the 6.7×10^{-5} threshold, meaning adversaries can easily generate inputs that fool verification for these layers. However, the output layer (L_3) exhibits significantly stronger resistance, with only a 5.1% pass rate at the same threshold, showing how reconstructing output layer activations from adversarial inputs is more challenging.

Note, however, our attacker must pass the separation threshold at *every* layer simultaneously, not just one of them. We can see the impact of this fact in Figure 10: The “All Layers” curve represents the percentage of attack attempts where all three layers simultaneously pass the threshold. Notably, this curve exactly matches the L_1 curve, revealing that L_1 acts as the bottleneck layer. This occurs because the adversarial optimization targeting the output layer creates a cascading effect where reconstruction difficulty increases for earlier layers, with L_1 consistently exhibiting the highest separation values. At the 10^{-3} threshold, while L_3 achieves a 78.6% pass rate, the “All Layers” rate matches L_1 at 0.0%.

7.3 Gradient Descent Attack on LLMs

Our second gradient-descent attack targets Llama-2-7B-chat-hf, treating the embedding layer as the input layer.

7.3.1 Setup

We implemented forward hooks across 290 layers spanning all 32 transformer blocks in Llama-2-7B-chat to capture intermediate values during both original inference and reconstruction attempts. The experimental samples consisted of 22 distinct text prompts covering various domains, each tokenized to a maximum sequence length of 10 tokens using the model’s native tokenizer with padding.

For each input, we performed 30 independent reconstruction attempts using Adam optimizer with learning rate 0.01 and L2 regularization ($\lambda = 0.001$), with up to 10,000 iterations per attempt or until convergence ($loss \leq 1e - 4$).

For each reconstruction, we evaluated 5,000 randomly selected neuron paths to measure model separation, plus one additional path where every neuron was tested and the one yielding the lowest path separation value was selected. This resulted in approximately 3.5M path evaluations across all experiments (25 inputs $\times$ 30 reconstructions $\times$ 5,001 paths

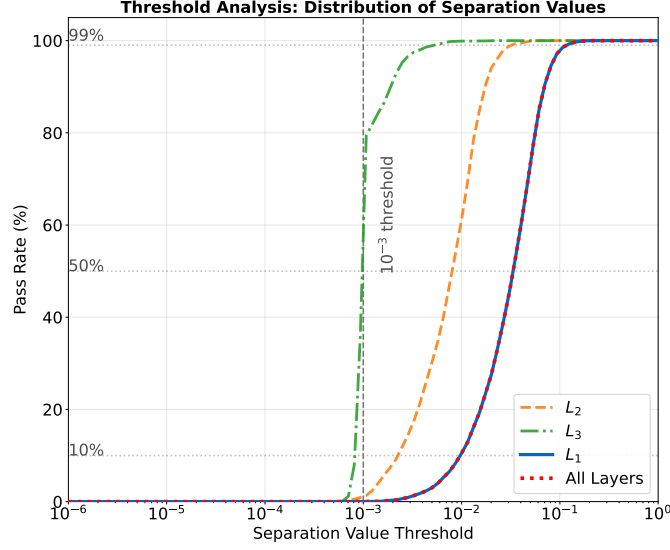


Figure 10: Pass rate analysis of mean separation values (log-scale). Individual layer curves (L_1, L_2, L_3) and the “All Layers” requirement show L_1 to be the bottleneck for attack success.

per reconstruction). This ensures statistical validity and demonstrates robust security against adversarial reconstruction even under relaxed time-complexity constraints.

7.3.2 Results

As shown in Figure 11, we measured the model separation between reconstructed activations and the true activations of the valid network. We find that across all 3.5M path evaluations, no reconstruction achieved separation values below 0.37, with a minimum observed separation value of 0.373. The cumulative distribution shows that the mean separation value is 0.410, the standard deviation is 0.010, and a median of 0.410, demonstrating that adversarial reconstruction efforts consistently failed to approximate the original activations.

Key percentiles further illustrate the robustness: the 1st percentile is 0.387, the 10th percentile is 0.396, the 25th percentile is 0.403, the 75th percentile is 0.418, and the 90th percentile reached 0.423. These results show that even when the adversary has the best possible conditions—including relaxed computational time and full model weights access—the model still has sufficient activation complexity to prevent successful activation reconstruction.

These findings demonstrate that despite relaxing computational constraints and providing complete model information, the inherent non-convexity of the optimization landscape and high-dimensional nature of transformer activations provide robust protection, with 100% of attempts failing to achieve meaningful reconstruction quality. Even the best-case reconstruction attempt maintained activation differences of 0.373, indicating substantial deviation from the original activations

8 Performance Evaluation

This section evaluates the performance of our verification protocol and compares it against zkLLM [SLZ24b], a state-of-the-art scheme based on cryptographic proofs tailored for

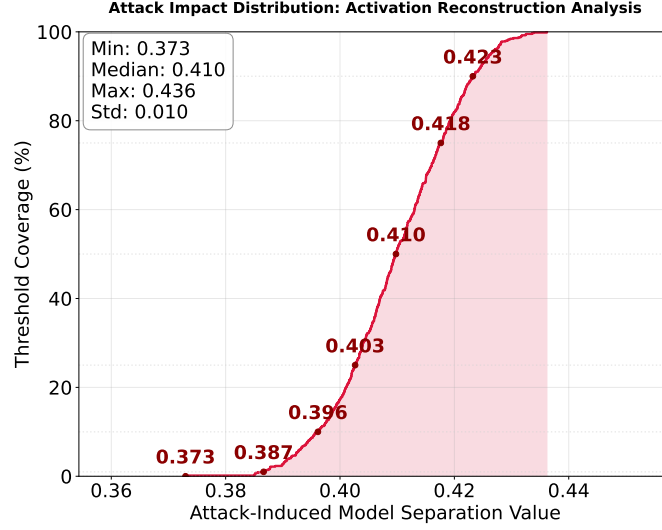


Figure 11: Model separation (Eq. 1) for Llama-2-7b-hf-chat at last token position, with neurons randomly selected per layer, and mean values aggregated per path. (x: separation value, y: ratio of tests below that threshold, red dots: percentile thresholds).

LLMs.

8.1 Evaluation Design and Setup

All experiments were conducted on a machine equipped with an AMD Ryzen 9 9900X CPU, 32GB of RAM, and an NVIDIA GeForce RTX 3090 GPU. Our evaluation focuses on the Llama-2-7B model. For zkLLM, we benchmarked the performance of their system using the publicly available implementation¹⁵ For our protocol, we measure the time to commit to the intermediate activation tensors of the network and the proof size. Specifically, our protocol generates a commitment to the trace by committing to the 192 intermediate tensors in the execution trace, each with dimensions of 64×4096 . To compute this efficiently, we use a hybrid commitment scheme that adapts well to the underlying architecture. This scheme uses a Merkle tree built on row-wise hashes (computed using SHA-256) but varies what it commits to:

- For matrix multiplication layers, where a non-learned input activation matrix is multiplied by a weight matrix, we commit only to the input activation matrix.
- For element-wise transformation layers, we commit directly to the output tensor.

This hybrid commitment scheme is key to our protocol’s high performance, as it does not serialize the entire computation. The proof contains the relevant row and the corresponding Merkle path for each layer in the network.

The zkLLM proof sizes were taken from their paper for the Llama-2-7B model. Although the hardware for the original zkLLM experiments differs, using our own timing benchmarks and their reported proof sizes provides a valuable baseline for understanding the relative performance of the two approaches.

8.2 Results

Our analysis reveals a significant performance advantage for our proposed protocol in terms of speed, with a trade-off in proof size. The performance of zkLLM on the Llama-2-7B

¹⁵Available at <https://github.com/jvhs0706/zkllm-ccs2024>.

model is as follows. The proving time for the entire inference process is 388.3 seconds. The proof size is 183 kB. The verification time is 2.36 seconds.

By switching from cryptographic assumptions to the separability assumptions demonstrated in Section 6, our protocol demonstrates a substantial improvement in efficiency. For proving time, the commitment and proof generation are remarkably fast. The time to commit to the entire trace is only 5.8 milliseconds. With the tensors in device memory, the proof generation for a single Merkle opening is on average 0.007 milliseconds. For the proof size, the base proof (the activation values of the neurons on the path) is 3.14 MB. This reduces to 1.6 MB using Brotli, a general purpose lossless compression algorithm. The Merkle openings (the cryptographic proof for the opened activations) add ≈ 100 KB to the total proof size, and this portion is not compressible. The total proof size is therefore ≈ 1.7 MB. Crucially, in our construction, the client does not possess the full model parameters locally but instead it holds a commitment to it. Therefore, the proof must also include the model weights and their Merkle openings. This effectively doubles the payload size. Consequently, the total proof size is approximately 3.4 MB. Our verification time is 12.44 milliseconds (two orders of magnitude lower).

9 Conclusion and Future Work

This work presents a novel and scalable verification framework for AI model inference that avoids the computational cost of producing a cryptographic proof for the full execution of a large neural network. A key conceptual contribution is a formal framework that allows protocol designers to leverage statistical properties of neural networks—specifically, trace separation between functionally dissimilar models—to argue the security of verifiable inference protocols. We instantiate this framework with a concrete protocol based on a “random spot checking paradigm”: the prover commits to the execution trace and opens only a small number of entries along randomly sampled paths from output to input. To validate our design (and the assumptions underpinning it), we designed and ran extensive experiments showing that models with different input/output behavior can be distinguished based on a few queries on their traces (list of all activation values).

Our experiments also provide evidence that our protocol resists natural attack strategies beyond the other-model soundness setting our framework formally captures. We designed and ran experiments showing that for the specific strategies we tested—gradient-descent reconstruction, inverse transforms, and logit swapping—it is still difficult to produce a “fake” trace that validates an incorrect output. Achieving full soundness, where the adversary may produce arbitrary outputs and forge traces without restriction, remains an open problem. In particular, any single-path sampling strategy faces an inherent detection bound of $1/N$ (where N is the maximum layer width), which may be limiting when the adversary can concentrate modifications in few nodes. Understanding how multi-path sampling and adaptive strategies can tighten this bound is an important direction for future work.

Additionally, we evaluated the proof size and time complexity of our verification protocol and showed that it greatly outperforms a state-of-the-art scheme for LLM verification in proving time, with moderately larger proofs.

Several promising avenues exist for future research. First, we welcome additional analysis of the protocol against more sophisticated and adaptive adversaries. For instance, a compelling adversarial strategy would be to train Generative Adversarial Networks to produce activation paths that are both internally consistent and statistically similar to honest traces. Additionally, exploring the security implications in applying the protocol to quantized models is important, as an adversary can exploit the rounding errors inherent in these lower precision architectures.

The concept of trace separation is widely agreed upon within the machine learning

community and our experiments provide evidence of this can be useful for verifiable inference. That being said, more formal theoretical guarantees of trace separation are needed to prove security. While work in formalizing the inner workings of neural networks is ongoing [MSM⁺25], the fact that no such theorem exists may be more of an indicator that such a result is non-trivial and will require future investigations.

Another key direction is expanding the applicability of the protocol to architectures with fundamentally different data flows, such as Graph Neural Networks (GNNs) [WPC⁺20]. Developing a secure non-interactive version of our scheme is an open problem: applying the common Fiat-Shamir heuristic to make the scheme non-interactive requires care to avoid known issues in “holographic” protocols like ours (based on accessing only a small part of the computation/inputs) and with non-negligible soundness error [CD24].

Finally, we plan to enhance the privacy of our protocol by using techniques from the cryptographic proofs literature (in a targeted manner) to avoid revealing raw activation values or potentially proprietary model parameters in the proofs. A naïve approach may be to simply apply a zero-knowledge proof on top of the verification scheme of our protocol, i.e., upon a path query by the verifier, the prover would answer with a zero-knowledge proof that the openings of the committed values satisfy the correct equations, without revealing the values. We expect this to still be faster than running the zero-knowledge proof directly on the entire network (as in zkLLM [SLZ24a]) since we are running it on a much smaller computation (i.e., the path). There are additional details to be considered, however, leaving an interesting space for future work.

10 Acknowledgments

The authors would like to thank Miranda Christ, Nelly Fazio, Ed Felten, Sanjam Garg, Nicola Greco, Mariana Raykova, Adi Shamir and Raúl Vicente for useful discussions. In particular, it was a conversation with Nicola Greco that planted the seed for the approach described in this paper.

This work was supported by Google under the Cyber NYC program. The views and conclusions contained herein are those of the authors and should not be interpreted as those of the sponsor.

References

- [AAF⁺19] Ulrich Aivodji, Hiromi Arai, Olivier Fortineau, Sébastien Gambs, Satoshi Hara, and Alain Tapp. Fairwashing: the risk of rationalization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 161–170. PMLR, 09–15 Jun 2019. URL: <https://proceedings.mlr.press/v97/aivodji19a.html>.
- [ACC⁺26] Pranay Anchuri, Matteo Campanelli, Paul Cesaretti, Rosario Gennaro, Tushar M. Jois, Hasan S. Kayman, and Tugce Ozdemir. Towards verifiable AI with lightweight cryptographic proofs of inference. 2026. To appear in 2026 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML).
- [AM12] Pablo Daniel Azar and Silvio Micali. Rational proofs. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1017–1028. ACM Press, May 2012. doi:10.1145/2213977.2214069.
- [Ano25] Anonymous. Uncensored AI in the wild: Tracking safety-modified open-weight LLMs. *Preprints.org*, 2025.

- [Arm21] Matthew Armstrong. Ethereum, smart contracts and the optimistic roll-up, 2021.
- [BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012. doi:10.1145/2090236.2090263.
- [BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. Cryptology ePrint Archive, Paper 2016/116, 2016. URL: <https://eprint.iacr.org/2016/116>.
- [CD24] Matteo Campanelli and Agni Datta. Fiat-Shamir goes rational. Cryptology ePrint Archive, Paper 2024/1645, 2024. URL: <https://eprint.iacr.org/2024/1645>.
- [CF13a] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *International Workshop on Public Key Cryptography*, pages 55–72. Springer, 2013.
- [CF13b] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Berlin, Heidelberg, February / March 2013. doi:10.1007/978-3-642-36362-7_5.
- [CG15] Matteo Campanelli and Rosario Gennaro. Sequentially composable rational proofs. In *Decision and Game Theory for Security (GameSec)*, volume 9406 of *LNCS*, pages 270–288. Springer, 2015. doi:10.1007/978-3-319-25594-1_15.
- [CG17] Matteo Campanelli and Rosario Gennaro. Efficient rational proofs for space bounded computations. In *Decision and Game Theory for Security (GameSec)*, volume 10575 of *LNCS*. Springer, 2017.
- [CGG24] Matteo Campanelli, Chaya Ganesh, and Rosario Gennaro. How to make rational arguments practical and extractable. *CiC*, 1(1):19, 2024. doi:10.62056/a63z186bm.
- [CHA24] Matteo Campanelli and Mathias Hall-Andersen. Fully succinct arguments over the integers from first principles. Cryptology ePrint Archive, Paper 2024/1548, 2024. URL: <https://eprint.iacr.org/2024/1548>.
- [CHM⁺19] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. Cryptology ePrint Archive, Paper 2019/1047, 2019. URL: <https://eprint.iacr.org/2019/1047>.
- [CRR13] Ran Canetti, Ben Riva, and Guy N Rothblum. Refereed delegation of computation. *Information and Computation*, 226:16–36, 2013.
- [CSYW24] KD Conway, Cathie So, Xiaohang Yu, and Kartik Wong. opML: Optimistic machine learning on blockchain, 2024. arXiv:2401.17555.
- [D⁺25] Yi Dong et al. Are you getting what you pay for? auditing model substitution in LLM APIs. *arXiv preprint arXiv:2504.04715*, 2025.
- [ea23] Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL: <https://arxiv.org/abs/2307.09288>, arXiv:2307.09288.

- [ea24] OpenAI et al. GPT-4 technical report, 2024. URL: <https://arxiv.org/abs/2303.08774>, [arXiv:2303.08774](https://arxiv.org/abs/2303.08774).
- [ea25a] DeepSeek-AI et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL: <https://arxiv.org/abs/2501.12948>, [arXiv:2501.12948](https://arxiv.org/abs/2501.12948).
- [ea25b] Gemini Team et al. Gemini: A family of highly capable multimodal models, 2025. URL: <https://arxiv.org/abs/2312.11805>, [arXiv:2312.11805](https://arxiv.org/abs/2312.11805).
- [FQZ⁺21] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. Zen: Efficient zero-knowledge proofs for neural networks. *IACR Cryptol. ePrint Arch.*, 2021:87, 2021.
- [GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.
- [Gol17] Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.
- [Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [JEP⁺21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.
- [KGC⁺18] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1353–1370, 2018.
- [KHSS22] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. Scaling up trustless dnn inference with zero-knowledge proofs. *arXiv preprint arXiv:2210.08674*, 2022.
- [Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992.
- [KMB08] Nikolaus Kriegeskorte, Marieke Mur, and Peter Bandettini. Representational similarity analysis—connecting the branches of systems neuroscience. *Frontiers in Systems Neuroscience*, 2:4, 2008. [doi:10.3389/neuro.06.004.2008](https://doi.org/10.3389/neuro.06.004.2008).

- [KNLH19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of Neural Network Representations Revisited. 5 2019. URL: <http://arxiv.org/abs/1905.00414>.
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [KSSL25] Max Klabunde, Tobias Schumacher, Markus Strohmaier, and Florian Lemmerich. Similarity of neural network models: A survey of functional and representational measures. *ACM Computing Surveys*, 57(9):1–52, May 2025. URL: <http://dx.doi.org/10.1145/3728458>, doi:10.1145/3728458.
- [LKKO24] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network based on zk-snarks. *IEEE Transactions on Dependable and Secure Computing*, 21(4):4254–4270, 2024.
- [LXZ21] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2968–2985, 2021.
- [Mer87] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.
- [Mer89] Ralph C Merkle. A certified digital signature. In *Advances in Cryptology—CRYPTO ’89: 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.
- [MSM⁺25] Giovanni Luca Marchetti, Vahid Shahverdi, Stefano Mereta, Matthew Trager, and Kathlén Kohn. Algebra unveils deep learning—an invitation to neuroalgebraic geometry. *arXiv preprint arXiv:2501.18915*, 2025.
- [Nit20] Anca Nitulescu. zk-SNARKs: A gentle introduction. *Ecole Normale Supérieure*, page 12, 2020.
- [OWJ⁺22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.
- [PHGR16] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.
- [QSL⁺25] Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. zkGPT: An efficient non-interactive zero-knowledge proof framework for llm inference. In *34th USENIX Security Symposium (USENIX Security 25)*, 2025.
- [QZX⁺23] Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! *arXiv preprint arXiv:2310.03693*, 2023.

- [RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 793–802. ACM, 2013.
- [Sin24] Shridhar Singh. Enhancing privacy and security in large-language models: a zero-knowledge proof approach. In *International Conference on Cyber Warfare and Security*, pages 574–582. Academic Conferences International Limited, 2024.
- [SLZ24a] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4405–4419, 2024.
- [SLZ24b] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 4405–4419. ACM Press, October 2024. doi:10.1145/3658644.3670334.
- [SLZ⁺25] Yifan Sun, Yuhang Li, Yue Zhang, Yuchen Jin, and Huan Zhang. Svip: Towards verifiable inference of open-source large language models, 2025. URL: <https://arxiv.org/abs/2410.22307>, arXiv:2410.22307.
- [Tha22] Justin Thaler. Measuring SNARK performance, November 2022. URL: <https://a16zcrypto.com/posts/article/measuring-snark-performance-frontends-backends-and-the-future>.
- [WH22] Haodi Wang and Thang Hoang. ezDPS: An efficient and zero-knowledge machine learning inference pipeline. *arXiv preprint arXiv:2212.05428*, 2022.
- [WPC⁺20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.
- [WYX⁺21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 501–518, 2021.
- [ZGC⁺25a] Xuandong Zhao, Sam Gunn, Miranda Christ, Jaiden Fairoze, Alec Fabrega, Nicholas Carlini, Sanjam Garg, Sanghyun Hong, Milad Nasr, Florian Tramèr, Somesh Jha, Lingxiao Li, Yu Wang, and Dawn Song. SoK: Watermarking for AI-generated content. In *IEEE Symposium on Security and Privacy*, pages 2621–2639, 2025.
- [ZGC⁺25b] Xuandong Zhao, Sam Gunn, Miranda Christ, Jaiden Fairoze, Andres Fabrega, Nicholas Carlini, Sanjam Garg, Sanghyun Hong, Milad Nasr, Florian Tramèr, Somesh Jha, Lei Li, Yu-Xiang Wang, and Dawn Song. SoK: Watermarking for ai-generated content. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, pages 2621–2639. IEEE, 2025.
- [ZLSJ22] Jingzhao Zhang, Haochuan Li, Suvrit Sra, and Ali Jadbabaie. Neural network weights do not converge to stationary points: An invariant measure perspective. In *International Conference on Machine Learning*, pages 26330–26346. PMLR, 2022.

A Motivating Scenarios for Other-Model Soundness

As discussed in Section 2, other-model soundness is a weaker security notion than full soundness: it only guarantees detection when the adversary runs a different model $\widetilde{\mathcal{M}}$ and commits to its real execution trace, rather than following an arbitrary cheating strategy. Yet, this restricted notion is practically useful. We argue that it captures several natural and pressing threat models, which we discuss through the following scenarios.

Economic incentives for serving outputs from different models. Hosting state-of-the-art large language models is expensive. This creates a natural economic incentive for providers to substitute the advertised model with a cheaper alternative (a smaller variant, a heavily quantized version, or simply a less capable model during peak demand [D⁺25]). Even when the intent is not malicious (e.g., load balancing during traffic spikes), undisclosed substitution breaks service agreements. It can also invalidate benchmarks that users rely upon when selecting a provider. Our protocol directly addresses this: if a provider commits to the weights of the advertised model $\mathcal{M}$ but serves inference from a different model $\widetilde{\mathcal{M}}$, verification will detect this with high probability.¹⁶

Safety alignment. Other-model soundness can also address threats to safety alignment. Modern LLMs undergo extensive alignment procedures to ensure they refuse harmful requests [OWJ⁺22]. However, recent work has shown that these safety measures are surprisingly brittle. Fine-tuning on as few as 100 examples can strip alignment from open-weight models [QZX⁺23], and the cost of doing so is minimal (under \$200 for a 70B-parameter model). Meanwhile, “uncensored” variants (models explicitly modified to remove safety constraints) are widely available on public repositories [Ano25].

A provider might substitute a safety-aligned model $\mathcal{M}$ with a dealigned variant $\widetilde{\mathcal{M}}$ to reduce user friction from refusals, or to handle requests the aligned model would decline. Since dealigned models produce outputs from fundamentally different distributions, our verification approach can detect such substitutions. This provides users and regulators with assurance that they are interacting with the certified, safety-aligned model rather than an unsafe substitute.

Training data compliance. In some settings model provenance matters legally or contractually. Consider a provider whose model $\mathcal{M}$ has been audited to certify that it was trained only on properly licensed data. The provider may be tempted to serve a different model $\widetilde{\mathcal{M}}$ if this alternative performs better. The latter may be trained on additional data, including material the provider had no right to use. This is not a hypothetical concern: the question of training data provenance is increasingly relevant in light of ongoing litigation over copyrighted content in training corpora [ZGC⁺25a]. Our notion of other-model soundness can capture this scenario.

B Formal Background on Vector Commitments

Informally, a vector commitment scheme maps a vector of length n of elements in some alphabet¹⁷ Σ . Formally, a vector commitment scheme over alphabet Σ consists of the following algorithms:

¹⁶This is true even if $\widetilde{\mathcal{M}}$ is a quantized or distilled version of $\mathcal{M}$, provided the two models produce sufficiently different output distributions on typical queries.

¹⁷By *alphabet* we refer to an arbitrary finite set. For simplicity, the reader can think of an alphabet as the set of binary strings of fixed length. However, an alphabet can be any convenient way to describe an object (in cryptography is occasionally common to use finite fields as alphabets, for example).

- $\text{GenParams}(1^\lambda, n)$: On input a security parameter λ and vector length n , outputs public parameters pp .
- $\text{CommitVec}(\text{pp}, v_1, \dots, v_n)$: Outputs a commitment cm to the vector $v \subseteq \Sigma^n$.
- $\text{Open}(\text{pp}, v, i)$: Generates a proof π_i for the value v_i at position i .
- $\text{Verify}(\text{pp}, \text{cm}, i, v_i, \pi_i)$: Returns 1 if π_i is a valid proof that $v_i \in \Sigma$ corresponds to the value in the committed vector at position i , and 0 otherwise.

A secure VC must satisfy *binding* (no efficient adversary can produce two different openings for the same index that both verify against the same commitment, except with negligible probability), *position binding* (no adversary can produce an accepting opening proof that claims a false value $\tilde{y} \neq v_i$ for position i against commitment $\text{cm} = \text{CommitVec}(\text{pp}, v)$). Efficient vector commitments are expected to have short commitments (often constant size), short proofs (polylogarithmic or constant size), and fast verification.

C Cryptographic Verifiable Inference Schemes

C.1 Basic Formal Definitions for Models

In order to be as precise as possible in our construction, we will define a very general abstract syntax and semantics for *family of models*. For the focus of this specific paper, the reader can think of families of models as *neural networks*; however, our notion is intentionally general in order to be potentially applicable in other settings.

Definition 5 (Family of models). A family of models over an alphabet Σ consists of a tuple $(n_{\text{model}}, n_{\text{qry}}, n_{\text{trc}}, \text{EvalTrace}, \text{Idxs}_{\text{out}}, f_{\text{out}})$ where:

- $n_{\text{model}}, n_{\text{qry}}, n_{\text{trc}} \in \mathbb{N}$ are positive integers denoting the size of the model, query and trace respectively (see also next items). Whenever we refer to an object as model, query or (evaluation) trace, we are referring to an element in the sets $\Sigma^{n_{\text{model}}}, \Sigma^{n_{\text{qry}}}, \Sigma^{n_{\text{trc}}}$ respectively.
- $\text{EvalTrace} : \Sigma^{n_{\text{model}}} \times \Sigma^{n_{\text{qry}}} \rightarrow \Sigma^{n_{\text{trc}}}$ is an efficiently computable deterministic function that associates to a model and a query a trace string. Given a model $\mathcal{M} \in \Sigma^{n_{\text{model}}}$ and a query $\text{qry} \in \Sigma^{n_{\text{qry}}}$, we refer to $\text{trc} := \text{EvalTrace}(\mathcal{M}, \text{qry})$ as the trace corresponding to the evaluation of $\mathcal{M}$ on qry (or evaluation trace for short). See also Remark 3 for some intuitions.
- $\text{Idxs}_{\text{out}} \subseteq \{1, 2, \dots, n_{\text{trc}}\}$ is a set of positions containing the outputs in an evaluation trace (see also definition of out below).
- $f_{\text{out}} : \Sigma^{|\text{Idxs}_{\text{out}}|} \rightarrow \{0, 1\}^*$ is a function that processes output-related parts of the trace—with indices Idxs_{out} —and returns the output (see also definition of out below and Remark 3).

Given a family of models, we define two additional notations that will be convenient to talk about outputs:

- **Output values in a trace string:** we define $\text{out}(\cdot)$ as the function that maps a trace trc (an element of $\Sigma^{n_{\text{trc}}}$) to its outputs. It is defined as some fixed function of some subset of the trace (e.g., some specific layer of the network). Explicitly,

$$\text{out}(\text{trc}) := f_{\text{out}}(\text{trc}_j)_{j \in \text{Idxs}_{\text{out}}}.$$

Notice that the verifier can compute this on their own by querying the appropriate elements of the trace, i.e., the positions in Idxs_{out} .

- **Output of a model run on a query:** given a model $\mathcal{M} \in \Sigma^{n_{\text{model}}}$ and a query $\text{qry} \in \Sigma^{n_{\text{qry}}}$, we denote by $\mathcal{M}(\text{qry})$ the actual output of the evaluation of $\mathcal{M}$ on query qry . That is, $\mathcal{M}(\text{qry}) := \text{out}(\text{EvalTrace}(\mathcal{M}, \text{qry}))$.

Remark 2 (Traces are binding to outputs). We will use the following straightforward observations to simplify the definition of Definition 1 (see also Remark 1). Consider a model $\mathcal{M} \in \Sigma^{n_{\text{model}}}$, a query $\text{qry} \in \Sigma^{n_{\text{qry}}}$ and an arbitrary trace string $\tilde{\text{trc}} \in \Sigma^{n_{\text{trc}}}$ (potentially provided by an adversary). Let $\text{trc} := \text{EvalTrace}(\mathcal{M}, \text{qry})$ be the genuine evaluation trace of $\mathcal{M}$ on qry . Then, if the adversary provides $\tilde{\text{trc}} \in \Sigma^{n_{\text{trc}}}$ (the alleged evaluation trace of $\mathcal{M}$ on qry), then it is also implicitly providing (and committing to) the alleged outputs of $\mathcal{M}$ on qry ; these are simply $\text{out}(\tilde{\text{trc}})$ and, by definition of out , can thus be simply retrieved by accessing¹⁸ the positions in $\tilde{\text{trc}}$ with indices in ldxs_{out} . If $\tilde{\text{trc}}$ is such that $\text{out}(\tilde{\text{trc}}) \neq \text{out}(\text{trc})$ then, obviously, by definition, $\text{out}(\tilde{\text{trc}}) \neq \mathcal{M}(\text{qry})$.

Remark 3 (Some examples and intuitions). For intuition, consider the case in which the family of models consists of feed-forward neural networks. Here, n_{qry} corresponds to (an encoding of) the input layer: a query $\text{qry} \in \Sigma^{n_{\text{qry}}}$ simply encodes the input features given to the network. Similarly, n_{model} encodes all network parameters (e.g., weights and biases flattened into a single string in $\Sigma^{n_{\text{model}}}$). The value n_{trc} naturally corresponds to the total number of neuron activations across all layers for a single forward pass.

Under this interpretation, $\text{EvalTrace}(\mathcal{M}, \text{qry})$ computes the usual forward pass of the neural network on input qry and returns a trace string containing all intermediate activations in a fixed order. Finally, $\text{ldxs}_{\text{out}} \subseteq \{1, \dots, n_{\text{trc}}\}$ is taken to be the set of indices corresponding to the output layer of the network, so that $\text{out}(\text{trc})$ recovers exactly the final-layer activations. Consequently, $\mathcal{M}(\text{qry}) = \text{out}(\text{EvalTrace}(\mathcal{M}, \text{qry}))$ is the standard notion of the network’s output on input qry .

For the case of an LLM, we could have ldxs_{out} correspond to the layer $\vec{z}$ of logit values and then define f_{out} as:

$$f_{\text{out}}(\vec{z}) := \text{softmax}(\vec{z}).$$

C.2 More On Our Design Choices

Model and Query Distributions. Our formalism describes models and queries as being sampled from distributions over them. We remark that this encompasses a broad range of situations. For example, in our definition of other-model soundness (Definition 1) it includes the case where there is a single honest model (the support of distribution $\mathcal{D}_{\text{model}}$ is a singleton, e.g., comprising the output of a model trained on a legal dataset) and we are concerned that the adversary might use for inference a model that has been trained on an illegal dataset (in this case the support of $\tilde{\mathcal{D}}_{\text{model}}$ will also be a singleton, including only the output of the training on the forbidden dataset¹⁹). Probability distributions over model are also expressive enough. For example, given a set $\mathcal{S}$ of “plausible” datasets, we could define $\mathcal{D}_{\text{model}}^{\mathcal{S}}$ so that sampling from that distribution, i.e., $\mathcal{M} \leftarrow_{\mathcal{S}} \mathcal{D}_{\text{model}}^{\mathcal{S}}$, essentially encodes “ $\mathcal{M}$ is any model with a specific architecture trained through a specific learning algorithm²⁰ and on any of the datasets in $\mathcal{S}$ ”. Having distributions on models and queries also permits *constraining* appropriately what are models and queries where we believe our assumptions required for security are true. Finally, we point out that we implicitly assume that all models we sample have the same architecture. This is without loss of generality: it is harder for the verifier to distinguish whether the adversary is using a particular model or not for inference when these have the same architecture. Therefore, whatever

¹⁸This type of random access is exactly the one that the client has in Definition 1 and that we enforce cryptographically (through vector commitments) in our construction/compiler in Section 4.4.

¹⁹We stress that the forbidden dataset may be adversarially chosen (as a function of the legal one) and may in principle overlap substantially with the legal dataset.

²⁰With specified parameters, such as learning rate and number of steps.

claim of security holds for the “same architecture” case, will hold for the “heterogeneous architecture” case.

Distance of Outputs. A “successful” adversary is intuitively one that is able to convince the client that the output of a model on a specific query is $\tilde{y}$, when instead it is some $y \neq \tilde{y}$. Our techniques will intuitively leverage the fact that the *more distant* an adversarial output is from the honest one, the *easier* it will be to detect it. For this reason we adopt a more general notion of “successful” adversary, one where the adversary may be required to provide not only $\tilde{y}$ distinct from the honest output, but one that is *sufficiently far* from it (above a certain threshold).

We point out that the security against “far enough” outputs is still of substantial applicability. An application may be robust with respect to “small deviations” from the honest output and thus not require security in that particular case. At the same time, there are intuitive ways in which supporting large deviations can be a useful security notion: depending on the distance definition, large deviations may for example capture the case of the adversary claiming an output in a completely different class (and this may be the type of false claims the application protects against). As an example, consider classification loss functions; a loss function using a soft target distribution (e.g., from word embeddings) penalizes mistakes less in cases where the predictions are of semantically similar class, while cross-entropy loss with one-hot encoding treats any misclassification as a large error.

D Our Protocol in the Refereed Model

D.1 Background

An interesting development in the verifiable computation is the refereed model of computation [CRR13], which addresses the problem of verifying a computation $y = F(x)$ in a setting where two parties, Prover P and Challenger V , make conflicting claims, $y_1 = F(x)$ and $y_2 = F(x)$, respectively. The model operates under the assumption that at least one party is honest. The objective is to enable an efficient, sublinear-time verification process by a third party, the Referee R , to determine which claim is valid.

The protocol leverages a challenge-response mechanism akin to a bisection on the computation’s execution trace: P provides an intermediate state S of the machine’s execution trace. If the state S is inconsistent with the prover’s claim, a discrepancy exists. V , assumed to be honest, identifies the segment of the computation—either from the initial state to S or from S to the final state—that is inconsistent with the correct execution. V then challenges P to reveal an intermediate state within the erroneous segment. This iterative bisection process continues until a single, incorrect computational step is isolated.

The protocol terminates after $O(\log T)$ rounds, where T is the number of steps in the computation of F . The final round requires the Referee to verify only a single, one-step computation between two adjacent states. The simplicity and high efficiency of this protocol have made it a cornerstone of optimistic rollup architectures (e.g., Arbitrum [KGC⁺18], Optimism [Arm21]), which accept off-chain computations as valid by default and only initiate a fraud-proof process—governed by this refereed model—in response to a challenge.

The application of the refereed model to verify the outputs of large-scale AI systems, such as deep neural networks, was initially proposed in [CSYW24]. While that work correctly identifies the inherent suitability of the refereed computation model for this domain—given the sublinear verification complexity and the often-extensive computational traces of large AI models—it does not provide empirical results or an experimental validation of this hypothesis.

D.2 Construction

In the refereed model we have two parties P_1, P_2 who make competing assertions about the output of a model $\mathcal{M}$ on input $\mathbf{qry}$. That is, P_i claims $\mathcal{M}(\mathbf{qry}) = y_i$ with $y_1 \neq y_2$. We assume that both P_1, P_2 know the full description of $\mathcal{M}$ and that the vector of weights defining $\mathcal{M}$ was honestly committed to via a vector commitment $C_{\mathcal{M}}$ available to all parties (both P_1 and P_2 know how to open $C_{\mathcal{M}}$ in order to provide proof of correctness for individual weights). As in our previous protocol, $C_{\mathcal{M}}$ serves as the “ground truth” about the model we are trying to verify.

Let n be the total number of nodes in model $\mathcal{M}$, and assume that the nodes are linearly ordered in a canonical way via topological sort. Without loss of generality we assume that the output of the model is the activation of the last node and the total number of nodes n is a power of 2 $n = 2^m$. Let H be a collision resistant hash function. The parties engage on the following protocol:

- P_i produces a commitment to a claimed hashed trace $C_i = \text{VC}(\text{htrc}_i)$. If P_i is honest it computes
 - $\text{trc}_i = \text{EvalTrace}(\mathcal{M}, \mathbf{qry}) = [a_{i,0}, \dots, a_{i,n}]$ the vector of activation values;
 - $A_{i,k} = [a_{i,k}, H(a_{i,0}, \dots, a_{i,k-1})]$ the pair of activation value $a_{i,k}$ and the hash of all the previous activation values;
 - $\text{htrc}_i = [A_{i,0}, \dots, A_{i,n}]$

Note that $A_{1,n} \neq A_{2,n}$ since the parties make competing assertions. Also note that there exists an index k_{in} such that if $k \leq k_{\text{in}}$, $a_{i,k}$ is an input node and therefore $A_{i,k}$ should match the input $\mathbf{qry}$.

- Set $k = 0$ and $\ell = n$
- While $\ell > k + 1$ do the following
 - set $u = (k + \ell)/2$
 - P_i opens $A_{i,k}, A_{i,\ell}, A_{i,u}$ w.r.t. C_i
 - if $k \leq k_{\text{in}}$ and $A_{i,k}$ does not match $\mathbf{qry}$ reject P_i 's claim and stop
 - if $A_{1,u} \neq A_{2,u}$ then $\ell \leftarrow u$ else $k \leftarrow u$
- If not stopped before, the loop ends with $a_{1,k} = a_{2,k}$ and $a_{i,\ell} \neq a_{2,\ell}$ with $\ell = k + 1$. The party P_i open $a_{i,j}$ and $w_{j,\ell}$ for $j \in G_\ell$, i.e., the activations and the weights of all the nodes that feed in node ℓ . Accept the claim of the party P_i for which $a_{i,\ell} = \sum_{j \in G_\ell} \phi(w_{j,\ell} a_{i,j})$.

Theorem 3. *If at least one of the two parties is honest the protocol always terminate by accepting the correct claim.*

Proof sketch. The claim is due to the following facts:

- The honest prover will never be caught inside the while loop;
- The protocol bisects through the internal nodes until it finds the *first* node (in the canonical order) in which the committed traces disagree (this is due to the collision resistance of the function H)
- Therefore in the last step only one of the two provers can survive the test (since $a_{1,j} = a_{2,j}$ for all the $j \in G_\ell$ since the topological sort guarantees that $j < \ell$).

□

Table 2: Percentage of Samples Passing Soundness Threshold by Method and Layer

Method	Layers	Threshold			
		10^{-4}	0.001	0.01	0.1
Pseudo Inverse	L_1	0.8%	9.2%	60.8%	100.0%
	L_2	0.0%	1.7%	17.5%	73.3%
	L_3	100.0%	100.0%	100.0%	100.0%
	All Layers	0.0%	0.0%	11.7%	73.3%
SVD	L_1	0.8%	9.2%	60.8%	100.0%
	L_2	0.0%	1.7%	17.5%	73.3%
	L_3	100.0%	100.0%	100.0%	100.0%
	All Layers	0.0%	0.0%	11.7%	73.3%
Regularized	L_1	0.0%	5.0%	62.5%	100.0%
	L_2	0.0%	1.7%	17.5%	73.3%
	L_3	100.0%	100.0%	100.0%	100.0%
	All Layers	0.0%	0.0%	12.5%	73.3%

E Inverse Transform Attack

E.1 Attack Design

Since the functions used in network computations are pseudo invertible, we utilize an inverse transform to explore the possibility that an adversary can adjust activations in a way that they will align with the claimed output. We used three different ways to invert matrices: The Moore-Penrose pseudo-inverse $\mathbf{W}^+ = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T$ for general least-squares solutions, the SVD-based inverse $\mathbf{W}^{-1} = \mathbf{V} \mathbf{S}^{-1} \mathbf{U}^T$ for numerical stability with ill-conditioned matrices, and the regularized inverse $\mathbf{W}^{-1} = (\mathbf{W}^T \mathbf{W} + \lambda \mathbf{I})^{-1} \mathbf{W}^T$ with $\lambda = 10^{-4}$ to deal with possible singularities.

With white-box access to model parameters, we apply matrix inversions one at a time, starting with the output layer and working our way back: First, we compute $\mathbf{a}^{(2)} = (\mathbf{W}^{(3)})^{-1} \mathbf{y}$, then $\mathbf{a}^{(1)} = (\mathbf{W}^{(2)})^{-1} \mathbf{a}^{(2)}$, and finally $\hat{\mathbf{x}} = (\mathbf{W}^{(1)})^{-1} \mathbf{a}^{(1)}$. On the ReLU with $h = \max(0, h)$ this attack was not successful; however for sigmoid-activated layers with $h \in [0, 1]$ it was. We study this attack on a simple ANN in Appendix E and note that this reconstruction attack approach for LLMs would be even more infeasible, due to its highly complex non-linearities (e.g., SiLU activations, skip-connection layers).

E.2 Experiment Setup

To evaluate the inverse transform attack, we utilize the same simple ANN architecture described in Section 7.2.1. We tested 120 input samples over 3 different inversion algorithms and compared reconstructed inputs and intermediate activations against ground truth values using absolute difference metrics (minimum, maximum, and mean) at each layer. The accuracy of the reconstruction is mostly limited by the difficulty of inverting certain activation functions and numerical precision limitation, and not because of computational intensity.

E.3 Results

Table 2 shows that the attack was unsuccessful. The output layer (L_3) was able to perfectly reconstruct all 120 test samples (100% passing even the strictest 10^{-4} threshold). However, the quality of the reconstruction got worse at deeper layers because of the cumulative effects of inverting activation functions. The most strict soundness criteria showed the attack’s flaws: at the 10^{-4} threshold, no samples were able to reconstruct all layers at the

same time for any inversion method. Only 0.8% of samples passed at L_1 and 0% at L_1 . However, relaxing the threshold to weaker levels create a more successful adversary: at the 0.1 threshold, 73.3% of samples achieved successful reconstruction across all layers simultaneously. In practice, note that, theoretically, our threshold could be 0, but to account for potential completeness concerns, we can set a reasonable 10^{-4} threshold—and even then, the adversary is unsuccessful.

The success of the reconstruction is heavily reliant on both the characteristics of the activation function and the structure of the network. For example, while invertible activation functions like sigmoid theoretically allow for inverse computation, our experiments utilized ReLU activations that cause irreversible information loss at negative values, which accounts for the observed degradation in reconstruction. Additionally, the dimensional relationships between layers—especially when a layer has more output neurons than input neurons—make systems that are overdetermined, which makes reconstruction more accurate even though these relationships are not linear. All three inversion methods yield remarkably consistent results, with only minor variations in the regularized approach at L_1 . This confirms that reconstruction quality is fundamentally limited by information loss at activation boundaries, numerical precision, and model architecture, rather than by the choice of inversion technique. The fact that reconstruction fidelity gets worse as depth increases—from perfect at the output layer to progressively worse—supports our analysis.

F Swap Attack on LLMs

F.1 Attack Design

The previous attacks test strong other-model soundness (Appendix H), where the adversary’s output comes from a specific substitute model. We now consider a more aggressive attack that is closer to full soundness: the adversary takes the true model’s output and directly perturbs it (by swapping the highest and lowest probability tokens), then tries to adjust intermediate activations to match. Unlike the previous attacks, no explicit substitute model is involved in this attack. The adversary constructs both the malicious output and the forged trace. Specifically, the swap attack employs a logit injection strategy in which the adversary first swaps the logit values of the highest- and lowest probability tokens to create a malicious target output and then optimizes intermediate activations to achieve this adversarial target while preserving the malicious logits as the desired output.

Specifically, we first compute the original model output: $\text{out} \leftarrow \mathcal{M}(\text{qry})$, and create malicious target logits $\mathbf{l}_{\tilde{\mathcal{M}}}$ by swapping the values at indices $\text{argmax}(\text{out})$ and $\text{argmin}(\text{out})$. We then initialize an optimizable activation tensor $a_{\tilde{\mathcal{M}}}$ from the output of the first block’s input normalization and inject it into the forward pass after the embedding layer. The adversary uses gradient descent to minimize:

$$L = (\text{out} - \text{out}_{\tilde{\mathcal{M}}})^2 + \lambda \|a^{(2)}\|^2$$

where $a_{\tilde{\mathcal{M}}}^{(2)}$ is updated as:

$$a_{\tilde{\mathcal{M}},i}^{(2)} = a_{\tilde{\mathcal{M}},i-1}^{(2)} - \alpha \frac{\partial L}{\partial a_{\tilde{\mathcal{M}},i-1}^{(2)}}$$

with α being the learning rate and $\lambda = 0.001$ for L2 regularization. During optimization, only the last token’s activations are updated (gradients for other positions are zeroed), and we iterate until convergence or up to 10,000 steps. This approach tests whether manipulating post-embedding, pre-transformer activations can successfully induce malicious model behavior.

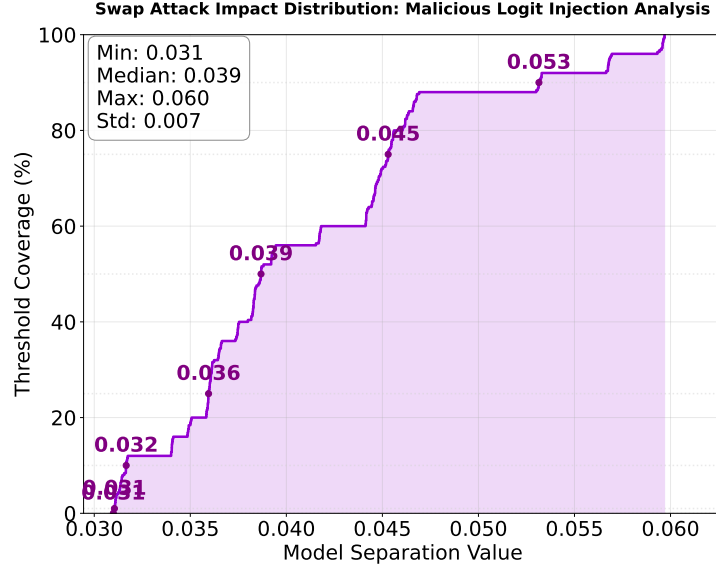


Figure 12: Model separation values (Equation (1)) for Llama-2-7b-hf-chat at last token position, with neurons randomly selected per layer and values aggregated by mean per path. Red dots mark key percentile points (x: separation value, y: percentage of tests below that threshold).

F.2 Experiment Setup

We used the same experimental setup as described in Section 7.3, with 290 forward hooks across all 32 transformer blocks capturing intermediate activations for 25 text prompts. However, unlike the activation reconstruction attack, the swap attack modifies the optimization objective and injection point.

For each input prompt, we first generated malicious target logits by swapping the highest and lowest probability output tokens. We then performed 10 independent reconstruction attempts, optimizing activations at the first transformer block’s input normalization layer (rather than at the embedding layer) using the same Adam optimizer configuration (learning rate 0.01, L2 regularization $\lambda = 0.001$) with up to 5,000 iterations or until convergence ($loss \leq 10^{-4}$).

Each reconstruction was evaluated on 5,001 paths (5,000 random + 1 optimized), yielding approximately 3.75M path evaluations ($25 \text{ inputs} \times 30 \text{ reconstructions} \times 5,001 \text{ paths}$) to measure separation values between original and adversarially modified activations.

F.3 Results

Our results, Figure 12, show the model separation values between adversarially-modified activations and the original activations under the logit swap attack.

We find that across evaluations, the swap attack achieved significantly lower separation values compared to direct activation reconstruction. The minimum observed separation value was 0.031, with a cumulative distribution showing a mean of 0.041, standard deviation of 0.010, and median of 0.039. Despite the adversary’s ability to directly manipulate intermediate activations and target specific malicious outputs through logit swapping, the separation values remained bounded.

Key percentiles illustrate the attack’s limited impact: the 1st percentile is 0.031, the 10th percentile is 0.032, the 25th percentile is 0.036, the 75th percentile is 0.045, and the 90th percentile reached 0.053. While these separation values are lower than those observed

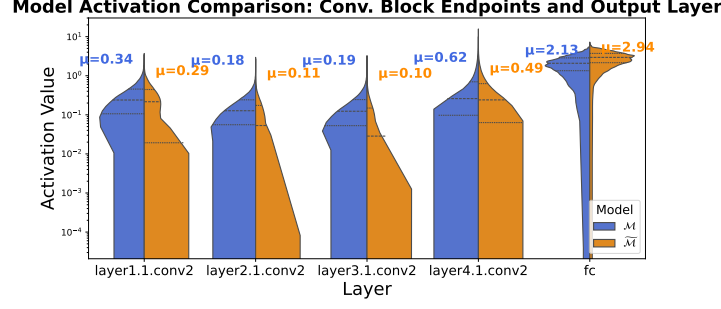


Figure 13: Activation distributions for path-selected neurons. Split violin plots show the probability density for neurons traversed during path selection across models trained on identical classification but different dog image subsets.

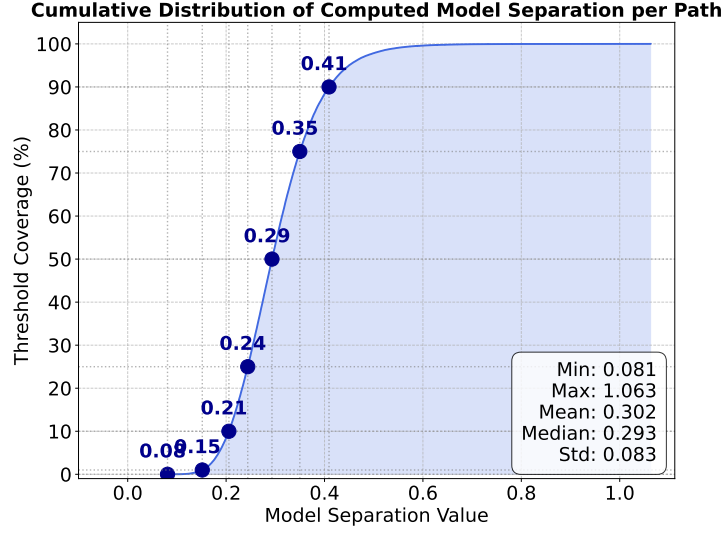


Figure 14: Cumulative distribution of model separation values. Shows separation magnitudes for path-selected neurons when both models perform dog vs cat classification with different dog training subsets.

in direct activation reconstruction (Section 7.3), they still demonstrate that the model maintains detectable differences between original and adversarially-manipulated activation paths. Even when the adversary can optimize activations at intermediate layers with full model access and relaxed computational constraints, the high-dimensional nature of transformer activations prevents perfect replication of the original forward pass.

These findings show that while manipulating post-embedding activations is more effective than reconstructing inputs from scratch, 100% of swap attack attempts still produced measurable activation deviations, with the best-case maintaining a separation value of 0.031 – an order of magnitude smaller than activation reconstruction but still indicating detectable differences in the activation space.

G Additional Model Separation Experiments

To ensure the robustness and generalization of our trace separation findings, we conducted two additional sets of experiments that study if model separation holds for classifiers

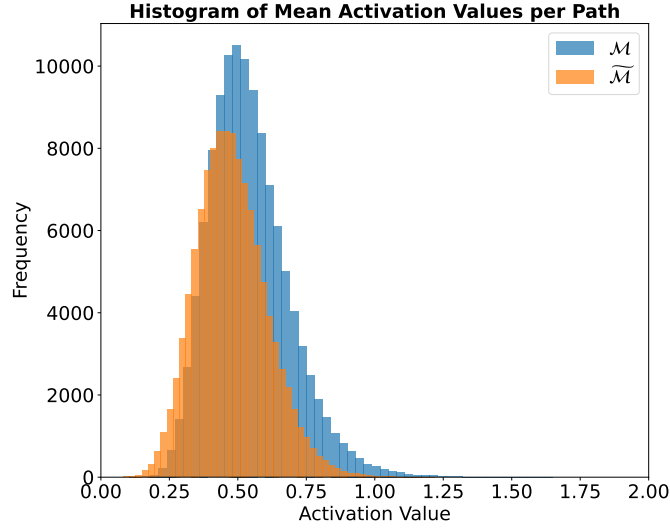


Figure 15: Mean activation values per path for dog vs cat classification with different dog training subsets. Each point represents the average activation across layers for one image-round pair.

trained on the same task and for classifiers trained at a higher granularity.

G.1 Model Separation in Classifiers for the Same Task

Our first extension is to see if we could measure trace separation on models trained on the same classification task. We replicated the experimental setup in Section 6.2 with two classifiers, but changed it as follows: we partitioned the dog dataset into disjoint subsets dog_A and dog_B (each with 1,500 images). $\mathcal{M}$ was given the entire cat dataset plus dog_A as its fine-tuning set, and $\widetilde{\mathcal{M}}$ was given the same cat dataset but with dog_B instead. So, both $\mathcal{M}$ and $\widetilde{\mathcal{M}}$ have the same classification task – distinguishing between cats and dogs – with a shared class but different subsets of data for the other task.

We reserved a pool of 2,509 validation images for trace separation tests. The minimum separation value, computed using Equation (1), is 0.081, with a mean of 0.302 and median of 0.293 across all paths. Notably, this is a higher degree of separation than shown in our initial experiment in Section 6.2.

The results demonstrated in Figure 13, the activation distributions for path-selected neurons, show similar patterns to our initial experiment in Section 6.2. The initial feature extraction layers (conv1) show virtually no difference (JS: 0), as both models learned comparable low-level features. However, the distributions expand substantially in deeper layers, with peak divergence occurring at layer3.1.conv2 (JS: 0.269). The final classifier layer (fc) maintains significant divergence (JS: 0.268), indicating that each model developed distinct decision boundaries despite solving identical tasks.

The cumulative distribution analysis (Figure 14) reveals that 25% of paths exhibit separation values below 0.244, while 75% remain below 0.350, demonstrating consistent measurable differences across the vast majority of inference paths. The mean activation histogram (Figure 15) confirms that these differences are clearly separable across the entire model. Thus, the results from this experiment demonstrate that even when models solve the same classification problem and share one class’s training data, differences in the other class’s training images induce measurable and consistent trace separation. Our results are in line with related results in understanding neural networks [ZLSJ22]. Indeed, our

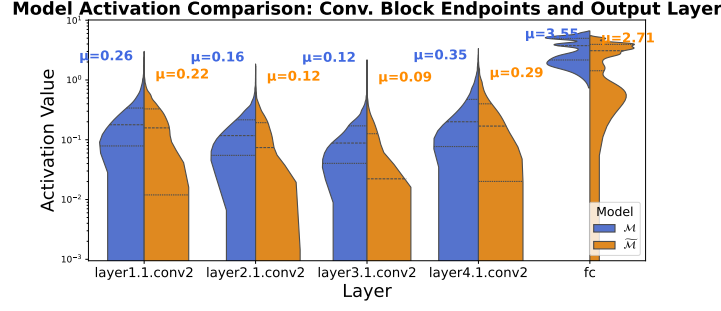


Figure 16: Activation distributions for path-selected neurons. Split violin plots show the probability density for neurons traversed during path selection across fine-grained dog breed classification models.

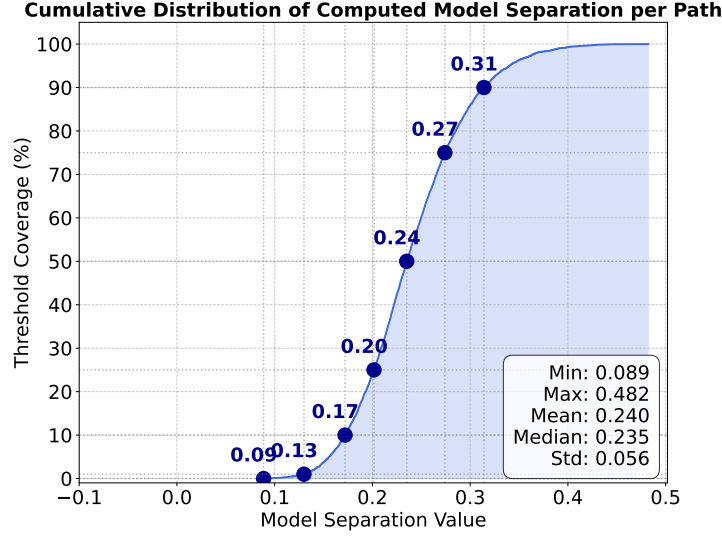


Figure 17: Cumulative distribution of model separation values. Shows separation magnitudes for path-selected neurons in fine-grained breed classification with Scottish Deerhound as the shared class.

results show that trace separation fundamentally captures dataset-level variations, instead of merely distinguishing classification objectives.

G.2 Model Separation in Classifiers for Granular Tasks

Our second experiment investigates if we can see trace separation for models trained on finer-grained characteristics than our dog-cat-squirrel experimental setup. So, instead of animal classification, we conducted dog breed classification using images from the Stanford Dogs Dataset.²¹ Specifically, $\mathcal{M}$ distinguished between Entlebucher and Scottish Deerhound breeds, while $\bar{\mathcal{M}}$ classified Bernese Mountain Dog versus Scottish Deerhound breeds. The shared Scottish Deerhound class enables comparison analogous to our trace separation setup in Section 6.2. Despite the visual similarity between these dog breeds, we observed robust trace separation across 26 test images evaluated over 250 rounds each (6,500 paths total), with minimum separation of 0.089, mean of 0.240, and median of 0.235.

²¹<http://vision.stanford.edu/aditya86/ImageNetDogs/>

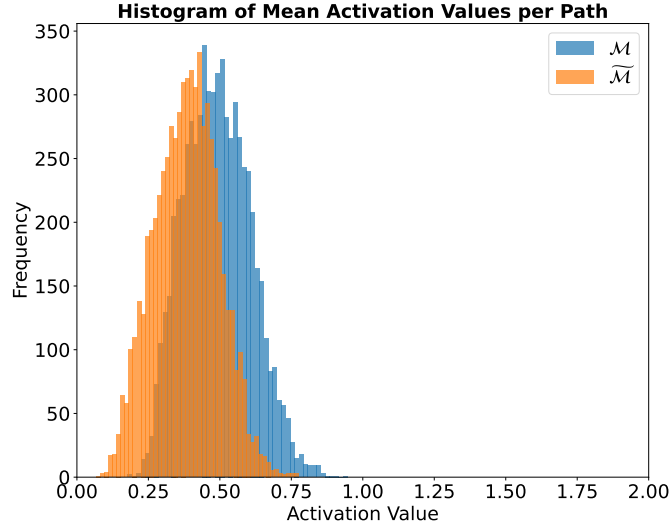


Figure 18: Mean activation values per path for fine-grained dog breed classification models. Each point represents the average activation across layers for one image-round pair in the breed classification experiment.

As shown in Figure 16, the activation distributions reveal minimal divergence in early layers (conv1: 0.000) but substantial separation in deeper layers, with peak divergence at layer3.1.conv2 (JS 0.287). Remarkably, the final classifier exhibited highest JS (0.719), indicating that even fine-grained classification tasks induce strong divergence in decision boundaries. The cumulative distribution (Figure 17) and mean activation analysis (Figure 18) confirm consistent separation patterns across all inference paths – even greater than the ones in Section 6.2. The fine-grained tasks clearly amplify separation, likely due the models developing highly structured decision boundaries. Our results in this experiment establish trace separation as a general property detectable even on finer-grained classification tasks.

H Strong Other-Model Soundness

In this section we define *strong other-model soundness*, a security notion that sits between other-model soundness and full soundness. Like other-model soundness, it restricts the adversary’s *output* to that of a specific alternative model $\widetilde{\mathcal{M}}$. However, unlike other-model soundness—where the adversary must commit to the honest evaluation trace of $\widetilde{\mathcal{M}}$ —strong other-model soundness allows the adversary to construct the trace *arbitrarily*, as in the full soundness setting.

This notion captures a natural threat scenario: an adversary who runs a different model to produce the output (see Appendix A), but who may attempt to craft a fake trace that passes the verification test (rather than simply submitting the honest trace of $\widetilde{\mathcal{M}}$). It is strictly stronger than other-model soundness (since the adversary has more freedom in choosing the trace) and strictly weaker than full soundness (since the adversary’s claimed output is constrained to $\widetilde{\mathcal{M}}(\text{qry})$ rather than being arbitrary).

H.1 Idealized Definition

We extend Definition 1 with the following additional soundness notion.

Strong other-model soundness. Let $\varepsilon : \mathbb{N} \rightarrow \mathbb{R}$ be a function and let $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ be as in Definition 1 and $\tilde{\mathcal{D}}_{\text{model}}$ be a model distribution (potentially distinct from $\mathcal{D}_{\text{model}}$). We say that the scheme satisfies (δ, ε) -strong-other-model-soundness w.r.t. $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ and $\tilde{\mathcal{D}}_{\text{model}}$ if for any adversary $\mathcal{A}$ and for any security parameter $\lambda \in \mathbb{N}$, we have:

$$\Pr \left[\begin{array}{l} \Delta[\mathcal{M}(\text{qry}), \tilde{\mathcal{M}}(\text{qry})] > \delta \wedge \\ \text{out}(\text{trc}) = \tilde{\mathcal{M}}(\text{qry}) \wedge \\ \text{TestTrace}^{\mathcal{M}, \text{trc}}(1^\lambda, \text{qry}) = 1 \end{array} : \begin{array}{l} \mathcal{M} \leftarrow_{\$} \mathcal{D}_{\text{model}} \\ \tilde{\mathcal{M}} \leftarrow_{\$} \tilde{\mathcal{D}}_{\text{model}} \\ \text{qry} \leftarrow_{\$} \mathcal{D}_{\text{query}} \\ \text{trc} \leftarrow \mathcal{A}(1^\lambda, \mathcal{M}, \text{qry}, \tilde{\mathcal{M}}) \end{array} \right] \leq \varepsilon(\lambda)$$

Informally, strong other-model soundness states that, except with ε probability, no adversary can fool the test into accepting a trace whose output matches that of a different model $\tilde{\mathcal{M}}$, even when the adversary is free to construct the trace in any way it chooses. The key difference from other-model soundness is in the last line of the sampling: the trace trc is produced by an adversary $\mathcal{A}$ (with access to $\mathcal{M}$, qry , and $\tilde{\mathcal{M}}$) rather than being the honest evaluation $\text{EvalTrace}(\tilde{\mathcal{M}}, \text{qry})$. The only constraint we impose for the adversary's output is that the trace it produces should be consistent with the output of the “wrong” model $\tilde{\mathcal{M}}$.

Remark 4 (Relation to other notions). The following implications hold:

- Full soundness implies strong other-model soundness: any adversary in the strong other-model soundness game can be simulated by a full soundness adversary that ignores the constraint on the output. More precisely, a full soundness adversary can sample $\tilde{\mathcal{M}} \leftarrow_{\$} \tilde{\mathcal{D}}_{\text{model}}$ on its own and run the strong-other-model-soundness adversary internally.
- Strong other-model soundness implies other-model soundness: the other-model soundness adversary is a special case where the trace is restricted to $\text{trc} = \text{EvalTrace}(\tilde{\mathcal{M}}, \text{qry})$.

H.2 Cryptographic Definition

We similarly extend Definition 4.

Strong other-model soundness. Let $\varepsilon : \mathbb{N} \rightarrow \mathbb{R}$ be a function and let $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ be as in Definition 4 and $\tilde{\mathcal{D}}_{\text{model}}$ be a model distribution (potentially distinct from $\mathcal{D}_{\text{model}}$). We say that the scheme satisfies (δ, ε) -strong-other-model-soundness w.r.t. $(\mathcal{D}_{\text{model}}, \mathcal{D}_{\text{query}}, \Delta)$ and $\tilde{\mathcal{D}}_{\text{model}}$ if for any efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and for any security parameter $\lambda \in \mathbb{N}$, we have:

$$\Pr \left[\begin{array}{l} \Delta[\mathcal{M}(\text{qry}), \tilde{\mathcal{M}}(\text{qry})] > \delta \wedge \\ \text{Verify}(\text{pp}, \text{cm}, \text{qry}, \tilde{y}, \\ (\pi_1, \pi_2), \rho) = 1 \end{array} : \begin{array}{l} \mathcal{M} \leftarrow_{\$} \mathcal{D}_{\text{model}} \\ \text{pp} \leftarrow \text{GenParams}(1^\lambda) \\ \text{cm} \leftarrow \text{CommitToModel}(\text{pp}, \mathcal{M}) \\ \tilde{\mathcal{M}} \leftarrow_{\$} \tilde{\mathcal{D}}_{\text{model}} \\ \text{qry} \leftarrow_{\$} \mathcal{D}_{\text{query}} \\ \tilde{y} := \tilde{\mathcal{M}}(\text{qry}) \\ (\pi_1, \text{state}) \leftarrow \mathcal{A}_1(\text{pp}, \mathcal{M}, \text{qry}, \tilde{\mathcal{M}}) \\ \rho \leftarrow_{\$} \mathcal{C} \\ \pi_2 \leftarrow \mathcal{A}_2(\text{state}, \rho) \end{array} \right] \leq \varepsilon(\lambda)$$

Note that, compared to the other-model soundness definition of Definition 4, the proof messages (π_1, π_2) are now produced by an adversary $\mathcal{A}$ rather than by the honest **Prove** algorithm run on $\tilde{\mathcal{M}}$. However, the claimed output $\tilde{y}$ remains fixed to $\tilde{\mathcal{M}}(\text{qry})$.

I Estimating thresholds and probabilities for other-model soundness

In this section, we describe a method for estimating the threshold values used in Definitions 2 and 3. The procedure estimates the *Trace Separation* parameters $(\delta_{sep}^{out}, \delta_{sep}^{trace})$ and validates the *Test Soundness* (ε_{tst}) . The approach we describe here works for both classification and LLM tasks.

Here, we present the simplified case where there is a single honest model $\mathcal{M}$ and a single adversarial model $\tilde{\mathcal{M}}$. In practice, the adversarial model is sampled from a distribution $\tilde{\mathcal{D}}_{\text{model}}$ (e.g., quantized models, pruned models, or specific adversarial fine-tunings)²². To ensure robust guarantees, the estimation procedure described below should be repeated on a diverse set of adversarial models $\{\tilde{\mathcal{M}}_1, \dots, \tilde{\mathcal{M}}_K\}$ sampled from $\tilde{\mathcal{D}}_{\text{model}}$. The final threshold parameters $(\delta_{sep}^{out}, \delta_{sep}^{trace})$ must be selected as the minimum observed separation values across all K ²³ instances to ensure that the **RandPathTest** is robust against worst case models.

I.1 Preliminaries and statistical layer selection

We start the estimation procedure by sampling a large set of N queries $\{\text{qry}_j\}_{j=1, \dots, N}$ and to generate the logit vectors and activation traces for both the honest model $\mathcal{M}$ and adversarial model $\tilde{\mathcal{M}}$.

1. **Statistical Layer Filtering:** We first compute the Jensen-Shannon (JS) divergence between the activation distributions of $\mathcal{M}$ and $\tilde{\mathcal{M}}$ for each layer. We define the set of valid layers $\mathcal{L}_{\text{valid}}$ as those where $JS > 0.05$, excluding noisy input layers. The JS threshold is chosen based on empirical observations where significant layers consistently exhibited divergence values well above this floor (e.g., > 0.09 in ResNet experiments, as detailed in Section 6.2).
2. **Metric 1: Output Difference ($dOut$).** Euclidean distance between logit vectors:

$$dOut(j) := \|\mathbf{y}_j - \tilde{\mathbf{y}}_j\|_2$$

3. **Metric 2: Trace Difference ($dTrc$).** Mean absolute difference over $\mathcal{L}_{\text{valid}}$. Let D_l be the dimension of layer l :

$$dTrc(j) := \frac{1}{|\mathcal{L}_{\text{valid}}|} \sum_{l \in \mathcal{L}_{\text{valid}}} \frac{1}{D_l} \|trc_{j,l} - \tilde{trc}_{j,l}\|_1$$

We end up with a set of N tuples

$$\mathcal{D} = \{(dOut(j), dTrc(j))\}_{j=1, \dots, N}$$

²²See also discussion in Appendix A.

²³ K represents the number of distinct variations in the adversarial model. It's set to a small number (e.g., 5-10).

I.2 Generating candidate thresholds

We first identify candidate pairs $\mathcal{C} \subseteq \mathcal{D}$ that satisfy the separation property with high confidence $(1 - \epsilon_{sep})$. Algorithm 1 takes as input the dataset we prepared from the queries and returns a subset of candidate threshold values for which the trace separation condition holds.

Algorithm 1 Generating Candidates given target ϵ_{sep}

```

1: Input: Dataset  $\mathcal{D} = \{(dOut(j), dTrc(j))\}_{j=1}^N$ , Target  $\epsilon_{sep}$  (e.g., 0.01), Noise Floor  $\delta_{noise}$  ( $10^{-4}$ )
2: Sort dataset by  $dOut(j)$ .
3: candidates  $\leftarrow \emptyset$ 
4: for each unique value  $x$  in sorted  $dOut$  do
5:   Let  $S_x$  be the subset of samples where  $dOut(j) \geq x$ 
6:   if  $|S_x|$  is too small then break
7:   end if
8:   Let  $q$  be the  $\epsilon_{sep}$ -quantile of  $dTrc$  values in  $S_x$ 
9:   if  $q > \delta_{noise}$  then
10:    Append pair  $(x, q)$  to candidates
11:   end if
12: end for
13: Output: List of candidates

```

I.3 Estimating path test soundness

We now describe a procedure that given a specific trace threshold δ estimates the probability that the path test fails to catch an incorrect trace that differs by at least δ . Algorithm 2 takes as input the dataset $\mathcal{D}$, a given trace separation value δ , and returns an estimate of the probability of failure of the procedure RandPathTest.

Algorithm 2 Estimating ϵ_{tst} for a fixed δ

```

1: Input: Dataset  $\mathcal{D}$ , Threshold  $\delta$ , Repetitions  $T$  (e.g., 50)
2:  $SumFailProb \leftarrow 0$ ,  $CountValid \leftarrow 0$ 
3: for  $j = 1, \dots, N$  do
4:   if  $dTrc(j) \geq \delta$  then ▷ Consider only traces with a difference  $\geq \delta$ 
5:      $CountValid \leftarrow CountValid + 1$ 
6:      $FailCount \leftarrow 0$ 
7:     for  $k = 1, \dots, T$  do
8:       Run RandPathTest on  $(\mathcal{M}, \tilde{trc}_j)$ 
9:       if Test Accepts (False Negative) then
10:         $FailCount \leftarrow FailCount + 1$ 
11:       end if
12:     end for
13:      $ProbFail_j \leftarrow FailCount/T$  ▷ Estimate for sample  $j$ 
14:      $SumFailProb \leftarrow SumFailProb + ProbFail_j$ 
15:   end if
16: end for
17:  $\hat{\epsilon}_{tst} \leftarrow SumFailProb/CountValid$  ▷ Overall estimate
18: Output:  $\hat{\epsilon}_{tst}$ 

```

I.4 Final parameter selection procedure

We now connect Algorithms 1 and 2 to select the optimal parameters. We seek the tightest Output Difference (δ_{out}) that enforces a Trace Difference (δ_{trace}) large enough for our RandPathTest test to detect reliably (low ϵ_{tst}).

1. Fix the maximum acceptable test error ϵ_{target} (e.g., 0.05).
2. Run **Algorithm 1** to get the list $\mathcal{C}$ of candidates.
3. Iterate through $\mathcal{C}$:
 - Let the current pair be $(\delta_{out}, \delta_{trace})$.
 - Run **Algorithm 2** using $\delta = \delta_{trace}$.
 - If the resulting $\hat{\epsilon}_{tst} \leq \epsilon_{target}$, select this pair and **Stop**.
4. **Result:** The selected $(\delta_{out}, \delta_{trace})$ are the final parameters. The protocol guarantees a soundness error of approximately $(\epsilon_{sep} + \epsilon_{tst})$.