Alon Rosen

Verifiable Inference and Training

The point is, Mr. Frampton, our viewers need proof.[*]

In *verifiable inference*, a prover aims to convince a verifier that $y = f_{\boldsymbol{\theta}}(x)$, while allowing the verifier to perform substantially less computation than evaluating $f_{\boldsymbol{\theta}}(x)$ itself.

For simple algebraic computations, randomness already gives some gain. Matrix multiplication can be verified using only matrix–vector products. For larger computations, a global claim can be reduced to a sequence of low-degree polynomial claims.

On arithmetic circuits, the resulting protocols give interactive proofs for inference and training. Succinct proof systems can further reduce communication and verifier work by relaxing soundness: a false statement may have an accepting proof, but no efficient prover should be able to find one. Zero knowledge is a separate property that can hide private inputs and intermediate values.

The main points of this chapter are:

- Randomness can make verifying a computation much cheaper than performing it.
- Large arithmetic computations can be turned into a sequence of small algebraic claims.
- Succinct proof systems trade information-theoretic soundness for computational soundness.
- Zero knowledge can hide private inputs and intermediate values.

[*]Monty Python, *Monty Python's Flying Circus*.

1 Verification

In *verifiable inference*, the prover should convince the verifier that $y = f_{\boldsymbol{\theta}}(x)$ while the verifier performs substantially less computation than evaluating $f_{\boldsymbol{\theta}}(x)$ itself.

A verification problem can be described by an NP relation $R(x, w)$, an efficiently checkable predicate where x is the public statement and w is a witness. The relation defines the language

$$L_R = \{x : \exists w,\ R(x, w) = 1\}.$$

The proposition to be proved is that $x \in L_R$. This gives a common language for inference, training, and other computation claims. For inference verification, the public statement is $(\boldsymbol{\theta}, x, y)$ and the witness is empty; the relation runs the inference computation and checks that $y = f_{\boldsymbol{\theta}}(x)$. If the prover wishes to keep the model private, the statement is instead (x, y) and $\boldsymbol{\theta}$ is the witness. We will later see how to prove such statements in *zero knowledge*, revealing no information beyond the fact that $y = f_{\boldsymbol{\theta}}(x)$ for some $\boldsymbol{\theta}$.

Verification is also relevant to *training*, where the computation is much longer. For simplicity, consider gradient descent

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L_{B_t}(\boldsymbol{\theta}_t), \qquad t = 0, \ldots, T-1,$$

where η is the learning rate, B_t is the set of training examples used at step t, and $L_{B_t}(\boldsymbol{\theta})$ is the loss on those examples. Here a natural NP witness can unroll all T updates and prove that $\boldsymbol{\theta}_T$ results from them, but for large models the trace is enormous.

Recomputing the result saves no verifier work. Having the prover send every value in an s-gate circuit does not help either: checking consistency and the output still costs $O(s)$. With *randomness* and *interaction*, the verifier can avoid inspecting the whole computation and use far less than $O(s)$ work. We also seek low *prover overhead*.

A proof of training may certify correct updates, use of committed data, final parameters, or completion of prescribed dependent steps. Yet it need not show meaningful learning: the model may start near its final state, make negligible progress, or be obtained another way and only later accompanied by a valid training trajectory. The proof establishes a valid computation, not that it was needed to obtain the model. Ruling out such shortcuts requires additional assumptions.

2 Interactive proofs

An interactive proof lets a randomized verifier check a statement by interacting with a prover.

Definition 1 (Interactive proof)**.** *Let R and L_R be as above. An interactive proof for R consists of a prover P and a randomized poly-time verifier V such that:*

- ***Completeness:*** *for every $x \in L_R$, $\Pr[\langle P, V\rangle(x) = 1] = 1$.*
- ***Soundness:*** *for every $x \notin L_R$ and every prover P^*, $\Pr[\langle P^*, V\rangle(x) = 1] \le \varepsilon$.*

Here ε is the *soundness error*. The probability is over the verifier's randomness and any randomness used by the prover. Independent repetition reduces the soundness error.

For deterministic computations, the claim is determined entirely by the public input, and we write $w = \perp$. A simple algebraic example that showcases the power of randomized verification is matrix multiplication.

Definition 2 (MM relation)**.** *For $A, B, C \in \mathbb{F}_q^{n\times n}$, define $R_{\mathrm{mm}}((A, B, C), \perp) = 1$ iff $AB = C$.*

Given a claimed product $C = AB$, the verifier samples $r \leftarrow \mathbb{F}_q^n$ uniformly and accepts if

$$A(Br) = Cr.$$

The standard algorithm uses about n^3 field operations, and asymptotically $O(n^\omega)$ with $\omega < 9/4$; verification uses $O(n^2)$. Even if $\omega = 2$, matrix–vector products may have much smaller constants, so verification can be faster in practice.

Proposition 1 (Randomized matrix-product verification)**.** *If $AB = C$, the verifier always accepts. If $AB \neq C$, it accepts with probability at most $1/q$.*

Proof. If $AB = C$, then $A(Br) = Cr$ for every $r \in \mathbb{F}_q^n$, so the verifier accepts with probability 1.

Suppose $AB \neq C$ and let $D = AB - C \neq 0$. Choose a nonzero row $d^\top$ of D. If the verifier accepts, then $Dr = 0$, and hence $d^\top r = 0$. Pick an index j with $d_j \neq 0$. After fixing all coordinates of r except r_j, there is exactly one value of $r_j \in \mathbb{F}_q$ for which $d^\top r = 0$. Therefore the verifier accepts with probability at most $1/q$. □

Over $\mathbb{F}_q$, k independent repetitions reduce the soundness error to at most q^{-k}. The reason the test works is that a nonzero low-degree polynomial cannot vanish at too many points. We will later see how to apply the same principle to more complicated algebraic identities.

3 From circuits to polynomials

To verify inference with much less work, we exploit algebraic structure. Arithmetic circuits over $\mathbb{F}_q$ have input, addition, and multiplication gates; matrix multiplication, bias addition, and polynomial approximations of nonlinearities fit directly.

Other operations need explicit constraints. For a ReLU $y = \max\{0, x\}$, encode bounded x by its sign and binary bits, then enforce $y = x$ for $x \geq 0$ and $y = 0$ otherwise. Quantized arithmetic represents real values by bounded integers at a fixed scale.

Assume the wiring predicates of each layer can be evaluated in $\operatorname{polylog} S$ time. For inference, the target is $y = f_{\boldsymbol{\theta}}(x)$ without checking every gate. Linear layers $z = Wu + b$ and convolutions fit the circuit model. The protocol developed below has $O(d \log m)$ interaction for circuit depth d and at most m gates per layer.

Consider a layer with at most 2^k gates. Number the gates by $b \in \{0,1\}^k$ and let $V(b) \in \mathbb{F}_q$ be the value at gate b. It is useful to replace $V : \{0,1\}^k \to \mathbb{F}_q$ by one polynomial defined on all of $\mathbb{F}_q^k$.

Definition 3 (Multilinear extension)**.** *For $V : \{0,1\}^k \to \mathbb{F}_q$, its multilinear extension is*

$$\widetilde{V}(z) = \sum_{b\in\{0,1\}^k} V(b)\chi_b(z), \qquad \chi_b(z) = \prod_{j=1}^{k} \big(b_j z_j + (1 - b_j)(1 - z_j)\big).$$

For every Boolean point b, $\chi_b(b) = 1$ and $\chi_{b'}(b) = 0$ for $b' \neq b$. Hence $\widetilde{V}(b) = V(b)$. Thus $\widetilde{V}$ agrees with V on the Boolean cube and can also be evaluated at arbitrary field points.

For a fixed circuit, define Boolean wiring predicates $\mathrm{add}(b, u, v)$ and $\mathrm{mult}(b, u, v)$, which indicate whether gate b takes its two inputs from gates u, v and applies addition or multiplication. Their multilinear extensions are low-degree polynomials known to the verifier.

At Boolean gate indices, with both sums over $u, v \in \{0,1\}^k$,

$$V_i(b) = \sum_{u,v} \mathrm{add}_i(b,u,v)\big(V_{i+1}(u) + V_{i+1}(v)\big) + \sum_{u,v} \mathrm{mult}_i(b,u,v) V_{i+1}(u) V_{i+1}(v).$$

This identity expresses an entire circuit layer as a low-degree sum over the next layer. Directly checking it would require summing over all pairs (u, v) of gates.

To verify a large circuit efficiently, we need a way to check such a sum without evaluating all of its terms. This is the role of the sumcheck protocol.

4 The sumcheck protocol

The circuit identity above leads to the following problem: verify a low-degree sum with exponentially many terms without evaluating them all.

Definition 4 (Sumcheck relation)**.** *Let $g : \mathbb{F}_q^m \to \mathbb{F}_q$ be a polynomial of degree at most d in each variable. Define $R_{\mathrm{sc}}((g, H), \perp) = 1$ iff $H = \sum_{b \in \{0,1\}^m} g(b)$.*

Verifying R_{sc} certifies that H is the sum of g over the Boolean cube without evaluating all 2^m terms. Such sums arise when checking the value of a circuit layer.

Theorem 1 (Sumcheck)**.** *There is an interactive proof for R_{sc} with perfect completeness and soundness error at most md/q.*

Proof. The protocol reduces the sum one variable at a time. The prover first sends the univariate polynomial

$$q_1(X) = \sum_{b_2,\ldots,b_m \in \{0,1\}} g(X, b_2, \ldots, b_m).$$

The verifier checks that $\deg(q_1) \le d$ and $q_1(0) + q_1(1) = H$. It then samples $r_1 \leftarrow \mathbb{F}_q$ uniformly and asks the prover to continue with

$$q_2(X) = \sum_{b_3,\ldots,b_m \in \{0,1\}} g(r_1, X, b_3, \ldots, b_m).$$

The consistency check is now $q_2(0) + q_2(1) = q_1(r_1)$. Continuing similarly for m rounds yields $r_1, \ldots, r_m$ and a final check of $g(r_1, \ldots, r_m)$. Thus the verifier receives m univariate polynomials of degree at most d and evaluates g once.

If $H = \sum_b g(b)$, an honest prover makes every consistency check and the final equality hold, so the verifier accepts with probability 1.

Suppose the initial claim is false. We use the elementary fact that a nonzero univariate polynomial of degree at most d over a field has at most d roots. At the first round in which the prover's polynomial differs from the polynomial determined by the current target value, the two degree-d polynomials can therefore agree at the verifier's fresh random challenge with probability at most d/q. Except for this event, the false claim is carried into the next round. A union bound over the m rounds gives soundness error at most md/q. □

Sumcheck reduces a sum over 2^m points to m low-degree consistency checks and one evaluation of g. For a circuit layer, this remaining evaluation can itself be rewritten using the next layer, allowing the reduction to continue.

Theorem 2 (Layered-circuit verification)**.** *Layered arithmetic circuits of size S have interactive proofs with near-linear prover time and* $\operatorname{polylog} S$ *verifier work per layer.*

Proof. Starting from the claimed output, the verifier maintains one claim $\widetilde{V}_i(r) = a$ at a random point r. Applying sumcheck to the circuit-layer identity reduces this to two evaluations of $\widetilde{V}_{i+1}$. The prover restricts $\widetilde{V}_{i+1}$ to the line through those two points; after checking the two endpoints, the verifier samples one fresh point on the line and keeps a single claim about layer $i + 1$.

Repeating reaches the input layer, where the verifier can evaluate the multilinear extension directly. Efficient wiring predicates give $\operatorname{polylog} S$ verifier work per layer, while the prover can evaluate the required sums with near-linear overhead over the circuit computation. □

5 Succinct arguments

Interactive proofs are sound against unbounded provers. Arguments relax this to *computational soundness*: only time-bounded provers must fail on false statements. Arguments can be shorter and faster to verify than proofs. Interaction can be removed using the *Fiat–Shamir* heuristic.

Definition 5 (Succinct argument)**.** *A (t, ε)-argument for a time-T verifiable relation R consists of a prover P and randomized poly-time verifier V such that:*

- ***Completeness:*** *for every $x \in L_R$,* $\Pr[\langle P, V\rangle(x) = 1] = 1$.
- ***Computational soundness:*** *for all $x \notin L_R$ and time-t provers P^*,* $\Pr[\langle P^*, V\rangle(x) = 1] \le \varepsilon$.

The argument is succinct *when communication and verifier work are* $\operatorname{poly}(|x|, \log T)$.

The cryptographic tool that enables the construction of succinct arguments is *collision-resistant hashing*, which can bind the prover to a short digest of a much larger object.

Definition 6 (Collision resistant hash)**.** *A hash distribution H is (t, ε)-collision resistant if, for $h \leftarrow H$, no time-t algorithm finds $x \neq x'$ with $h(x) = h(x')$ with probability $> \varepsilon$.*

The construction encodes the computation as a probabilistically checkable proof (PCP) Π, as guaranteed for NP by the PCP theorem. Since the verifier reads only a few locations, the prover first commits to Π by placing its blocks in a binary hash tree and sending the root digest. To open a queried block, it sends the block and sibling hashes along its path to the root, using $O(\log |\Pi|)$ hashes. Collision resistance prevents inconsistent openings, so the verifier queries one fixed PCP.

Theorem 3 (Succinct arguments)**.** *Assuming collision-resistant hash functions, every NP relation has a succinct argument with communication and verifier work* $\operatorname{poly}(|x|, \log T)$.

For an input of length n, the PCP has polynomial length, while its verifier uses $O(\log n)$ random bits and reads only $O(1)$ bits of Π, with perfect completeness and soundness at most $1/2$. Repetition reduces the soundness error to ε using $O(\log(1/\varepsilon))$ queries.

A succinct argument need not have an efficient prover: succinctness bounds communication and verifier work, not proof generation. If checking a computation takes time T (for example, when evaluating a model or checking a training trace), the verifier can use $\operatorname{poly}(|x|, \log T)$ work. The prover must still construct the PCP and hash tree; generic PCP construction may take $T^{O(1)}$ time.

The construction is *public coin*: the verifier sends fresh random challenges, independent of secret state. The *Fiat–Shamir* heuristic makes such protocols non-interactive by deriving challenges from

the public statement and preceding transcript. For example, after prover messages $m_1, \ldots, m_i$, set $r_i = H(x, m_1, \ldots, m_i)$. The prover then generates the transcript alone; the verifier recomputes the challenges and checks it. Soundness is usually analyzed with H modeled as a random oracle; an arbitrary deterministic hash need not preserve soundness.

An alternative is a *common reference string* (CRS): public parameters generated during setup and shared by both parties. A trusted setup can yield short proofs and fast verification, amortizing its cost over many proofs. Security depends on correct setup, and some schemes require new parameters when the computation changes.

6 Zero knowledge

For public input x and any efficient verifier V^*, let $(P, V^*)(x)$ denote the distribution of its view: its own randomness and the prover messages it receives. Zero knowledge requires this view to be efficiently simulated from x alone.

Definition 7 (Zero knowledge)**.** *An interactive proof (P, V) for L is computational zero knowledge if every efficient V^* has an efficient randomized simulator S such that, for all $x \in L$,*

$$(P, V^*)(x) \approx_c S(x).$$

Thus, an efficient verifier gains no information from the interaction beyond what it could generate from the public statement: its real and simulated views are computationally indistinguishable.

Definition 8 (Inference relation)**.** *For (x, y) and witness $\boldsymbol{\theta}$, $R_{\text{inf}}((x, y), \boldsymbol{\theta}) = 1$ iff $y = f_{\boldsymbol{\theta}}(x)$.*

This relation is already quite general: if $f_{\boldsymbol{\theta}}$ ranges over polynomial-size circuits, the witness $\boldsymbol{\theta}$ can encode arbitrary auxiliary data and computation, so R_{inf} can represent general NP statements.

Theorem 4 (Zero knowledge for NP)**.** *If one-way functions exist, every language in NP has a computational zero-knowledge interactive proof.*

The idea is to reduce any given NP language to 3-coloring, which is NP-complete. From the public statement x, both parties compute a graph G; from an NP witness w, the prover also obtains a proper 3-coloring of G. Thus it suffices to prove that G is 3-colorable.

- The prover randomly permutes the three colors and commits to the color of every vertex.
- The verifier chooses a random edge; the prover opens its endpoint colors, which must differ.
- Commitments hide unopened colors, and the random permutation hides the coloring.

A simulator guesses the challenged edge and rewinds if necessary. Because hiding commitments conceal their contents much like encryption, the simulation is computationally indistinguishable from a real interaction even if the simulator commits to garbage away from the challenged edge.

A zero-knowledge proof for R_{inf} reveals no information beyond what the public statement and the fact that $y = f_{\boldsymbol{\theta}}(x)$ for some $\boldsymbol{\theta}$ already reveal. However, as we saw in Chapter 7 on model extraction, black-box access *can* leak model information.

If the model must be fixed in advance, the prover can publish a commitment c to $\boldsymbol{\theta}$, which hides the model while binding the prover to it. The proof checks both $R_{\text{inf}}((x, y), \boldsymbol{\theta}) = 1$ and consistency with c. For training, the witness may instead contain the data and intermediate optimization states.

For modern ML, prover overhead remains a central obstacle: generic NP reductions can add substantial arithmetization and cryptographic cost despite good asymptotics.

Zero knowledge does not by itself hide the circuit: a public circuit reveals the architecture, layer dimensions, and operations. Hiding this structure requires a universal circuit or program representation, with additional overhead.

7 Historical notes

Freivalds introduced randomized matrix-product verification [1]. Lund et al. developed sumcheck [2]; Goldwasser et al. combined it with multilinear extensions for layered circuits [3].

Goldwasser et al. introduced interactive proofs and zero knowledge [4]; Goldreich et al. proved zero knowledge for NP from one-way functions [5]. Kilian built succinct arguments from PCPs and collision-resistant hash trees [6]; Micali developed computationally sound proofs [7]. Fiat and Shamir replaced random challenges with transcript hashes [8].

For ML, zkCNN proves convolutional inference [9], and Balbás et al. compose layer-specific protocols via modular sumcheck [10]. Garg et al. study proofs of training [11]; Kaizen scales them via recursion [12].

References

[1] R. Freivalds. *Fast Probabilistic Algorithms*. Mathematical Foundations of Computer Science, 1979.

[2] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. *Algebraic Methods for Interactive Proof Systems*. Journal of the ACM 39(4), 1992.

[3] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. *Delegating Computation: Interactive Proofs for Muggles*. Journal of the ACM 62(4), 2015.

[4] S. Goldwasser, S. Micali, and C. Rackoff. *The Knowledge Complexity of Interactive Proof Systems*. SIAM Journal on Computing 18(1), 1989.

[5] O. Goldreich, S. Micali, and A. Wigderson. *Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems*. Journal of the ACM 38(3), 1991.

[6] J. Kilian. *A Note on Efficient Zero-Knowledge Proofs and Arguments*. STOC, 1992.

[7] S. Micali. *Computationally Sound Proofs*. SIAM Journal on Computing 30(4), 2000.

[8] A. Fiat and A. Shamir. *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*. CRYPTO, 1986.

[9] T. Liu, X. Xie, and Y. Zhang. *zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy*. ACM CCS, 2021.

[10] D. Balbás, D. Fiore, M. I. González Vasco, D. Robissout, and C. Soriente. *Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing*. ACM CCS, 2023.

[11] S. Garg et al. *Experimenting with Zero-Knowledge Proofs of Training*. IACR ePrint 2023/1345.

[12] M. Abbaszadeh, C. Pappas, J. Katz, and D. Papadopoulos. *Kaizen: Zero-Knowledge Proofs of Training for Deep Neural Networks*. IACR ePrint 2024/162.