Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Verifiable Deep Learning: Smart Contract--Enabled Model Integrity and Provenance on Distributed Ledgers

Faguo Wu
Beihang University
Bo Yuan
Beijing Academy of Science and Technology
Yiou Wang
wangyiou90@163.com

Beihang University

Research Article

Keywords:

Posted Date: November 3rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11377517/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Verifiable Deep Learning: Smart Contract–Enabled Model Integrity and Provenance on Distributed Ledgers

Anonymous Author[1,2*], Anonymous Author[2,3†] and Anonymous Author[1,2†]

[1*]Department of [Anonymous], [Anonymous University], [Anonymous], [Anonymous], 000000, [Anonymous], [Anonymous].
[2]Department of [Anonymous], [Anonymous University], [Anonymous], [Anonymous], 000000, [Anonymous], [Anonymous].
[3]Department of [Anonymous], [Anonymous University], [Anonymous], [Anonymous], 000000, [Anonymous], [Anonymous].

*Corresponding author(s). E-mail(s): anonymous@review.com;
Contributing authors: anonymous@review.com; anonymous@review.com;
†These authors contributed equally to this work.

Abstract

Deep learning artifacts undergo long and mutable lifecycles in which datasets, preprocessing pipelines, checkpoints, evaluation configurations, and deployed versions may be controlled by different actors. Hash anchoring makes individual bytes tamper-evident but does not establish that a checkpoint was derived from authorized data or that its alleged computation actually occurred. We present ProofWeave, a provenance-coupled verification specification that treats model evolution as a typed, policy-constrained transition system. Each transition binds content commitments, authenticated parent references, version rules, and explicit evidence obligations; the proposed contract state machine admits only policy-valid transitions. We distinguish full computational proofs, conditioned on a sound external proving backend, from probabilistic committed-step audits with a finite-population detection bound. For transparent evaluation we implement a *local integrity prototype*, not a live blockchain or zero-knowledge proof service. Seven deterministic integrity fault types are assessed across three public datasets; four nested validators cover 2, 4, 6, and 7 of the nine broader threat classes discussed. To expose an essential security boundary, we additionally perform 105 real MLP training runs over seven random seeds and five controlled training-label

corruption rates. Hash/signature microbenchmarks and Merkle trees of up to 65,536 leaves characterize local scaling. A controlled Monte Carlo audit study quantifies probabilistic detection under committed-step sampling, separately from the local measurements. The evidence supports preliminary provenance-check correctness and quantifies its limits; it does not establish on-chain costs or end-to-end verifiable neural training.

Keywords: trustworthy machine learning; model provenance; distributed ledger; smart contract; verifiable computation; cryptographic commitments; auditability

1 Introduction

Deep learning models are increasingly developed and deployed through complex workflows involving data collection, preprocessing, training, evaluation, and model updates. In collaborative environments, the organizations using a model may not have direct access to its training data, intermediate checkpoints, or execution records. This makes it difficult to verify whether a delivered model was produced through the claimed process. Existing studies have examined the licensing and attribution of training datasets [1], while subsequent research has highlighted problems related to data consent and authenticity [2, 3]. Model transparency studies have also emphasized the importance of documenting development practices and model-related information [4, 5]. However, such documentation does not itself verify the computational history of a model. A cryptographic hash can confirm that an artifact has not changed, but it cannot establish whether the artifact was generated from authorized data through the declared training procedure.

Existing research has explored blockchain, cryptographic commitments, and zero-knowledge proofs to improve the verifiability of machine learning systems. Blockchain-assisted learning frameworks record data and model-update information to support integrity checking and provenance tracking [6, 7]. Related approaches introduce verifiable mechanisms for collaborative and federated learning [8, 9]. In parallel, zero-knowledge machine learning has enabled the verification of neural network inference without revealing private inputs or model information [10, 11]. Recent systems further extend these techniques to large language model inference [12] and neural network training [13]. Commit-and-prove constructions provide additional mechanisms for linking computational proofs to existing commitments [14]. Despite these advances, verifying individual artifacts or computations does not automatically establish the integrity of the entire model development process, particularly when datasets, model versions, and training operations are managed by different parties.

Three challenges remain in establishing end-to-end model integrity and provenance. First, verifying an individual checkpoint is insufficient when its validity also depends on the authenticity of its training data, preprocessing operations, and preceding model versions. Existing proof-of-training research has investigated how to verify learning computations [15, 16], but these guarantees must also be connected to the corresponding data and model histories. Second, different verification mechanisms provide different

levels of assurance. Full computational proofs can provide strong guarantees for precisely specified operations, but their computational costs remain substantial [17, 18]. Lightweight auditing offers a more practical alternative in some settings, although checking only selected operations cannot guarantee the correctness of an entire training process. Third, blockchain-based verification must coordinate these mechanisms while enforcing consistent authorization, versioning, and provenance rules. These challenges motivate a unified verification framework that connects model artifacts, computational evidence, and lifecycle policies within a common model-evolution structure.

We propose ProofWeave(Figure 1), a *proof-carrying model-evolution graph* governed by a transition-validity invariant. Each data, preprocessing, training, evaluation, or deployment operation emits a typed transition record binding the output commitment to an explicitly ordered set of parent commitments and to an authorization policy. An off-chain prover or auditor produces evidence against the transition relation; the smart contract checks the corresponding verification result and advances only valid graph states. The same semantics support strong full-computation proofs when a suitable proof backend is present and a weaker but quantifiable post-commitment challenge mode when complete proof generation is impractical. We contribute:

1. **Provenance-coupled verification semantics.** We formalize a typed transition relation linking artifact commitments, authenticated ancestry, declared operation semantics, and admission policies instead of verifying disconnected hashes.
2. **Evidence-aware smart-contract transition design.** We specify a deterministic ledger state machine with distinct FULL-PROOF and CHALLENGE obligations, anti-replay constraints, and composable verification receipts.
3. **Conditional security analysis.** We derive a lineage-integrity reduction under stated cryptographic and consensus assumptions and an exact finite-population miss probability for sampling committed faulty transitions.
4. **Reproducible initial evaluation.** We report repeated-seed label-corruption sensitivity, targeted mutation checks, payload-size latency, six-scale Merkle costs, and a Monte Carlo audit study, isolating detectable provenance faults from signed semantic manipulation.

2 Related Work

2.1 Verifiable neural computation

Early systems such as SafetyNets and zkCNN laid a foundation for verifiable neural inference [19, 20]. More recent methods investigate proof generation for non-linear layers, convolutional networks, and large models [10–12, 17, 18, 21–25]. Proof-of-training methods shift attention from final inference to the integrity of optimization histories, including their computational cost and random-seed assumptions [13, 15, 16, 26–28]. Surveys detail both the promise and resource bottlenecks of these approaches [29, 30]. Our focus is orthogonal to replacing their proof kernels: we specify the public statement and model-lifecycle linkage that any chosen proof kernel must authenticate.

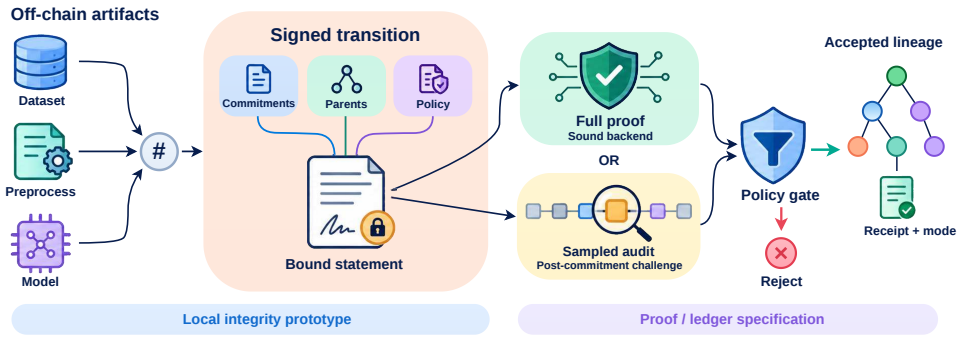


Fig. 1 Proposed ProofWeave architecture. A signed transition binds artifact commitments, parent references, and policy to a common statement. Full proofs or post-commitment audits support policy-gated admission, producing an accepted lineage and a mode-labeled receipt. The local prototype implements integrity and lineage checks; the proof backend and ledger remain architectural specifications.

2.2 Cryptographic commitment and incremental verification

Proof systems including Spartan and PLONK provide the basis for succinct relation checking [31, 32]. Modern lookup arguments and incremental/folding constructions reduce the cost of repeated verification [33–38]. Artemis directly addresses the often-overlooked gap between proof witnesses and previously registered external commitments [14]. Consequently, ProofWeave distinguishes a correct computation over some witness from a correct computation *over the committed lineage witness*.

2.3 Distributed provenance and accountability

Blockchain-assisted federated or decentralized learning registers model updates, protects contribution accounting, and arbitrates untrusted participants [6–9, 39–41]. Further studies address accountable data sharing, authenticated off-chain results, IoT-model provenance, and combined ZKP/blockchain validity checks [42–46]. However, a signed or ledger-anchored assertion is not evidence of the computational truth of that assertion. Our formalization makes the strength of the evidence obligation an explicit part of every admitted state transition. Dataset documentation and model-card proposals supply valuable complementary human-interpretable information but do not replace computational proof [1, 4, 47, 48].

3 Methodology

3.1 Problem formulation and trust boundaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed acyclic graph of model-lifecycle artifacts. Each vertex $v \in \mathcal{V}$ corresponds to an operation type

$$\tau_v \in \{\mathsf{Acquire}, \mathsf{Preprocess}, \mathsf{Train}, \mathsf{Evaluate}, \mathsf{Deploy}\}. \tag{1}$$

The vertex references its ordered parent identifiers $\mathbf{p}_v = (\mathrm{id}_{u_1}, \ldots, \mathrm{id}_{u_m})$, an artifact a_v, a version ν_v, a policy identifier κ_v, and an authenticated issuer s_v. We model the operation as a transition $a_v = F_{\tau_v}(a_{u_1}, \ldots, a_{u_m}; \omega_v)$ with executable parameters and randomness ω_v. Different execution environments may be nondeterministic; therefore an exact execution claim must fix the bit-level semantics of F_{τ_v} or adopt a carefully specified tolerance relation. A provenance claim is separate from a quality, legality, or adversarial-robustness claim.

We assume (i) a collision-resistant hash and unforgeable signature scheme; (ii) a ledger with the stated deterministic execution and finality properties; (iii) correct admission policies; (iv) an external proof backend with soundness whenever full proof is requested. We do *not* assume that an opaque hash proves training correctness, that a signature proves the signer is honest, or that the ledger certifies the accuracy of model predictions.

3.2 Canonical commitments and transition identity

We represent tensors with an agreed ordering, shape, dtype, endianness, finite-value convention, serialization version, and hashing domain. For a canonical serializer ser and domain tag `ART`, define

$$d_v = \mathsf{H}\big(\texttt{ART} \parallel \mathrm{ser}(a_v)\big), \tag{2}$$

$$b_v = \mathsf{H}\big(\texttt{META} \parallel \tau_v \parallel d_v \parallel \kappa_v \parallel \nu_v \parallel \mathrm{ser}(\omega_v)\big), \tag{3}$$

$$\mathrm{id}_v = \mathsf{H}\Big(\texttt{NODE} \parallel b_v \parallel \mathrm{ser}(\mathbf{p}_v) \parallel s_v\Big). \tag{4}$$

A transaction signs the complete public manifest $m_v = (\tau_v, d_v, \mathbf{p}_v, \kappa_v, \nu_v, s_v, \ldots)$:

$$\sigma_v \leftarrow \mathsf{Sig.Sign}_{sk_{s_v}}(\mathsf{H}(\texttt{SIGN} \parallel \mathrm{ser}(m_v))). \tag{5}$$

Distinct tags prevent accidental cross-context reuse of hashes; ordered parent commitments prevent silent parent permutation. A large input set can be summarized with a binary Merkle root r_v, where leaf hash $h_i = \mathsf{H}(\texttt{LEAF} \parallel a_i)$ and internal hash $h_{i,j} = \mathsf{H}(\texttt{NODE} \parallel h_i \parallel h_j)$. Membership proofs require $O(\log n)$ hashes and grow as $32\lceil \log_2 n \rceil$ bytes for SHA-256 sibling paths, excluding direction metadata.

3.3 The provenance-coupled execution relation

Let x_v be a public statement containing the input/output commitments, transition type, policy root, authorized parent set, version, and proof-verification key. Let the private witness w_v contain the underlying tensors, private execution arguments, and any auxiliary opening information. Rather than separately validating a training certificate and an unrelated lineage digest, we define the *joint* relation

$$\mathcal{R}_{\tau_v}(x_v, w_v) = \mathcal{R}_{\mathrm{bind}} \wedge \mathcal{R}_{\mathrm{lineage}} \wedge \mathcal{R}_{\mathrm{policy}} \wedge \mathcal{R}_{\mathrm{exec}}, \tag{6}$$

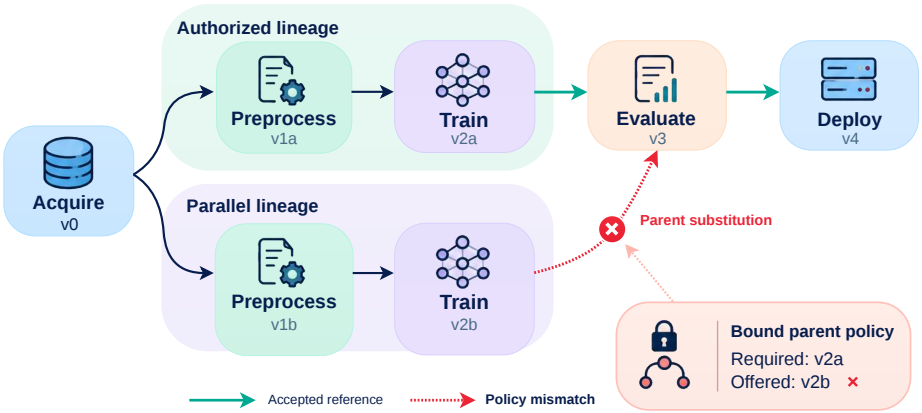


Fig. 2 Typed graph semantics. Two model lineages share an acquisition node. Evaluation is bound to parent v2a and feeds deployment. Substituting v2b is rejected when the authenticated parent reference or the required role/version policy fails.

where the individual predicates are

$$\mathcal{R}_{\text{bind}} = \big[d_v = \mathsf{H}(\text{ser}(a_v))\big] \wedge \bigwedge_{u \in \text{Pa}(v)} \big[d_u = \mathsf{H}(\text{ser}(a_u))\big], \tag{7}$$

$$\mathcal{R}_{\text{lineage}} = \big[\text{Pa}(v) \subseteq \mathcal{V}_{\text{accepted}}\big] \wedge \big[\text{TypeAllowed}(\tau_v, \tau_u)\big], \tag{8}$$

$$\mathcal{R}_{\text{policy}} = \big[\text{Auth}(s_v, \tau_v, \kappa_v) = 1\big] \wedge \big[\nu_v > \max_{u \in \text{Pa}(v)} \nu_u\big], \tag{9}$$

$$\mathcal{R}_{\text{exec}} = \big[a_v = F_{\tau_v}(\mathbf{a}_{\text{Pa}(v)}; \omega_v)\big]. \tag{10}$$

Domain separation was suppressed in Eq. (10) for readability; Eq. (4) is normative. When a valid SNARK/zkVM/IVC backend supports the declared F_{τ_v}, a computational witness is proved as

$$\begin{aligned} \pi_v &\leftarrow \mathsf{Prove}(pk_{\tau_v}, x_v, w_v), \\ z_v &= \mathsf{Verify}(vk_{\tau_v}, x_v, \pi_v) \in \{0, 1\}. \end{aligned} \tag{11}$$

The *public statement must cryptographically bind the commitments in Eq.* (4); otherwise a genuine proof may concern an unrelated model. A full proof establishes the encoded computation relation up to the soundness error and does not by itself show that the committed dataset was lawful, representative, or free of poison. Figure 2 illustrates the typed lifecycle graph and a parent substitution that violates the bound admission policy.

3.4 Evidence obligations and transition policies

We specify a proof obligation $o_v = (\texttt{mode}_v, q_v, \kappa_v, \text{vkid}_v)$. In FULL-PROOF mode, acceptance requires $z_v = 1$. In CHALLENGE mode, the issuer first commits to a fixed trace $T_v = (t_1, \ldots, t_N)$, and an unpredictable challenge chosen after commitment selects k trace locations for deterministic recomputation or validated external certificates. The mode is explicitly recorded; the resulting verdict cannot be promoted to a universal execution proof.

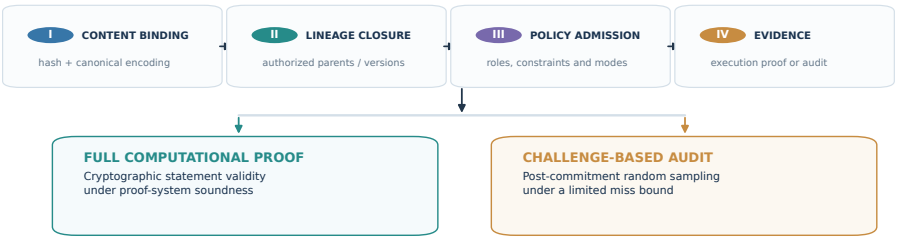


Fig. 3 Single transition-validity invariant. The three gates share one public statement, preventing mismatches between committed lineage and proved computation. The two evidence modes yield different claims and must not be conflated.

For optional workload-aware auditing, a risk score $\rho_v \in \mathbb{R}$ may influence sampling while a protocol-enforced floor remains in place:

$$q_v = \max\left\{q_{\min},\ \mathrm{sigmoid}(\alpha\rho_v + \beta)\right\}, \tag{12}$$

$$k_v = \max\left\{k_{\min}, \lceil N_v q_v \rceil\right\}. \tag{13}$$

A malicious actor might manipulate the risk score; therefore only $k_{\min}$ under a trusted post-commitment challenge is security-critical. For uniform sampling without replacement, exactly b faulty steps among N fixed committed steps evade all k checks with probability

$$\Pr[\text{miss} \mid N, b, k] = \frac{\binom{N-b}{k}}{\binom{N}{k}} \le \left(1 - \frac{b}{N}\right)^k. \tag{14}$$

Equation (14) applies only if sampled checks detect the selected type of fault and the adversary cannot rewrite the committed trace after challenge generation. It does not apply to undetectable semantic corruption or nondeterministic recomputation without a declared tolerance test.

3.5 Smart-contract state transition

The ledger maintains $_t = (\mathcal{V}_t, \mathrm{Policy}_t, \mathrm{Nonce}_t, \mathrm{Receipt}_t)$. For each submitted tuple (m_v, σ_v, π_v), admission is defined by

$$\begin{aligned} A_v^{\mathrm{meta}} = {} & \mathrm{SigValid}(m_v, \sigma_v) \wedge \mathrm{ParentsAccepted}(v) \\ & \wedge \mathrm{TypesValid}(v) \wedge \mathrm{PolicyValid}(v) \\ & \wedge \mathrm{Fresh}(v), \end{aligned} \tag{15}$$

$$A_v^{\mathrm{evid}} = \begin{cases} \mathsf{Verify}(vk_{\tau_v}, x_v, \pi_v), & \texttt{mode} = \textsc{full-proof}, \\ \mathrm{CheckChallengeReceipt}(v), & \texttt{mode} = \textsc{challenge}, \end{cases} \tag{16}$$

$$_{t+1} = \begin{cases} _t \cup \{v\}, & A_v^{\mathrm{meta}} \wedge A_v^{\mathrm{evid}}, \\ _t, & \text{otherwise}. \end{cases} \tag{17}$$

Algorithm 1 Provenance-coupled transition admission

Require: Ledger , manifest m, signature σ, evidence e

if $\neg$VerifySignature(m, σ) **then**
  return REJECT
end if
if $\neg$CanonicalHash(m) or $\neg$ParentsAccepted(m) **then**
  return REJECT
end if
if $\neg$AllowedType(m) or $\neg$FreshVersion(m) **then**
  return REJECT
end if
if $\neg$AuthorizedPolicy(m) **then**
  return REJECT
end if
if m.mode = FULL **then**
  $b \leftarrow$ VerifyComputationProof(m, e)
else
  $b \leftarrow$ ValidatePostCommitmentChallenge(m, e)
end if
if $\neg b$ **then**
  return REJECT
end if
$\leftarrow$ AppendImmutableReceipt$(, m, e)$
return ACCEPT

A smart contract must verify the evidence or a trust-scoped authenticated verifier result according to its declared policy. Merely storing a proof hash or third-party statement does not check that proof. The latter case should be labeled *attested verifier mode*, with the verifier's trust assumption disclosed. A chain reorganization beyond the assumed finality window invalidates the permanence interpretation of earlier commitments.

3.6 Composed correctness and claim levels

Suppose a published model v has m relevant ancestors $\mathrm{Anc}(v)$. A deployment certificate verifies the ancestry path and every required proof obligation:

$$\mathrm{ValidDeploy}(v) = \bigwedge_{u \in \mathrm{Anc}(v) \cup \{v\}} \left(A_u^{\mathrm{meta}} \wedge A_u^{\mathrm{evid}} \right). \tag{18}$$

This is a *provenance-closure* condition: an accepted deployment requires all mandatory parent transitions to remain accepted under their originally bound policies. If a training operation is authorized under CHALLENGE mode, the resulting deployment cannot be described as fully proven even when later inference steps use full proofs.

Assume canonical encodings, collision resistance of H, EUF-CMA security of the signature scheme, a correctly executed immutable ledger state machine, and knowledge soundness of the selected full-computation proof backend. For any polynomial-time

adversary attempting to obtain acceptance of a transition with a modified committed artifact, unauthorized parent, or false full-computation statement, the success probability is bounded by

$$\Pr[\mathsf{BadAccept}] \leq \epsilon_{\mathrm{coll}} + \epsilon_{\mathrm{sig}} + \epsilon_{\mathrm{policy}} + \sum_{u \in \mathrm{Anc}(v) \cup \{v\}} \epsilon_{\mathrm{proof},u}, \tag{19}$$

where $\epsilon_{\mathrm{policy}}$ denotes violations of policy/ledger correctness assumptions. For sampled transitions, Eq. (14) additionally bounds failures to detect the specified committed-step faults, rather than proving all steps correct.

Proof sketch. A modified artifact preserving its commitment yields a hash collision. A modified authenticated manifest without a valid issuer yields a signature forgery. An unauthorized ancestry or replay that survives a correctly enforced typed-policy predicate contradicts state-machine correctness. A false computed statement with accepted full evidence violates proof-system soundness. A union bound over the accepted ancestry yields Eq. (19). Challenge mode has distinct sampling semantics and therefore cannot be inserted as a zero-error term. □

Cost model.

For M lifecycle transitions, storing a constant-size commitment and bounded metadata per transition requires $O(M)$ ledger records, independent of model tensor size, while raw artifacts remain off-chain. Merkle proof verification costs $O(\log n)$ hashes; full proving cost is determined by the selected circuit/backend and may dominate. A simple latency decomposition is

$$T_{\mathrm{end}} = T_{\mathrm{encode}} + T_{\mathrm{commit}} + T_{\mathrm{prove/audit}} + T_{\mathrm{contract}} + T_{\mathrm{consensus}}, \tag{20}$$

where this paper measures only selected local terms, not T_{prove}, T_{contract}, or $T_{\mathrm{consensus}}$.

4 Experimental Evaluation

4.1 Scope, data, and evidence labels

The evaluation comprises three complementary components: local execution measurements from classifier training and cryptographic operations; deterministic verification checks and their coverage analysis; and a controlled Monte Carlo study of random trace challenges. These components use different evaluation targets and are interpreted separately. No results in this section come from a deployed EVM/Fabric contract, a live validator network, an implemented zk-SNARK, or an authenticated training-proof backend. ProofWeave is therefore experimentally a *local provenance-validator prototype*, while the ledger and proof systems remain a formal specification.

Public datasets distributed with scikit-learn are Digits (1,797 samples, 64 features, 10 classes), Breast Cancer Wisconsin (569 samples, 30 features, two classes), and Wine (178 samples, 13 features, three classes). All classification trials share a stratified 75/25 split (random state 42); a training-fitted StandardScaler is applied to both splits. We train one-hidden-layer MLP classifiers with 48 hidden units, $\alpha = 10^{-3}$, a cap of 180

Table 1 Test accuracy (%, mean ± standard deviation over seven random seeds) under different training-label corruption rates. All datasets use a 75/25 train–test split.

Dataset	n	Features	Classes	0%	5%	10%	20%	30%
Digits	1,797	64	10	95.2 ± 1.2	94.8 ± 1.3	93.7 ± 2.0	92.5 ± 1.4	89.9 ± 2.9
Breast cancer	569	30	2	93.1 ± 2.6	94.1 ± 1.9	91.7 ± 1.9	91.3 ± 4.4	90.4 ± 3.1
Wine	178	13	3	83.8 ± 7.0	72.7 ± 17.5	80.0 ± 9.1	59.4 ± 21.5	60.6 ± 24.0

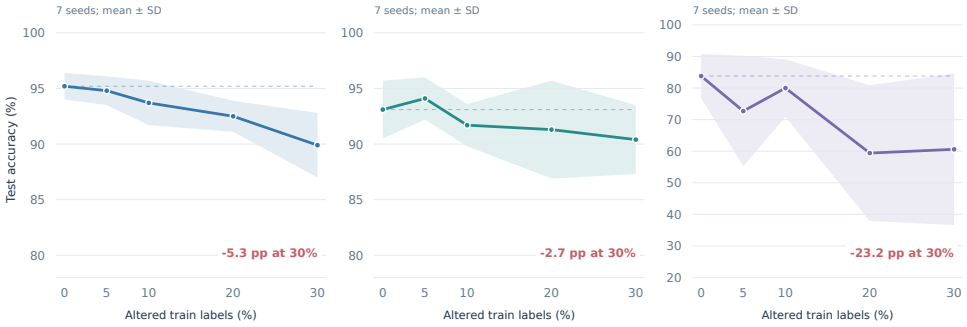


Fig. 4 Label-perturbation sensitivity. Seven-seed mean accuracy and ±1 standard deviation on held-out clean labels for each public dataset. The label-flip intervention changes authorized training data and is not an integrity fault detectable from a correct commitment alone.

optimizer iterations, and early stopping. Seven independently seeded initializations (11, 23, 37, 51, 79, 101, 137) are tested at five training-label flip rates (0, 5, 10, 20, 30%). At nonzero rates we uniformly select training examples and replace each original class with a distinct class. Thus **105 classifier training runs** were executed; the held-out test labels are never modified.

4.2 RQ1: Can a valid signature coexist with a degraded model?

Table 1 and Fig. 4 report measured test accuracy across repeated seeds and training-label perturbations. The signatures and content hashes in the proposed provenance design authenticate *whatever input was actually committed*; they do not independently decide whether these inputs are clinically, legally, or statistically legitimate. Consequently, if the corrupted training labels are committed by an authorized signer, the local validator can consistently approve the record while model quality changes. At 30% altered labels, Digits falls from **95.2% to 89.9%** mean accuracy and Wine from **83.8% to 60.6%**. Breast Cancer decreases from **93.1% to 90.4%**; the 5% case may improve slightly from sampling variance. Standard deviations, particularly on the small Wine dataset, are important to the interpretation. This is a measured *limitation of provenance-only assurance*, not an accuracy improvement attributed to ProofWeave.

Table 2 Threat taxonomy and verification scope. P1–P7 are directly exercised by deterministic local fault injection; S1–S2 are identified security gaps, not measured proof failures. "Rejected" refers only to the declared metadata predicates, never to a live contract.

ID	Injected scenario / security condition	Required discriminator	Local full-check outcome
P1	Weights modified after signing	Hash mismatch	Rejected
P2	Wrong committed artifact digest	Hash mismatch	Rejected
P3	Unsigned manifest version edit	Signature mismatch	Rejected
P4	Corrupted signature bytes	Signature mismatch	Rejected
P5	Signed but unknown parent	Typed ancestry	Rejected
P6	Signed stale version replay	Version monotonicity	Rejected
P7	Signed unauthorized policy	Policy identity	Rejected
S1	Authorized, signed label-poisoned training set	Semantic data quality	Not detected
S2	Authorized, signed but false training claim	Computation proof missing	Not detected

4.3 RQ2: What failure classes can metadata validation actually reject?

Table 2 expands the fault taxonomy to nine classes. The first seven are **measured deterministic local checks**: for each of the three datasets, 100 mutations of each kind were injected into a signed training transition and checked against the local reference verifier ($3 \times 7 \times 100 = 2{,}100$ check invocations). The full checker rejects the targeted seven cases by construction. This is best interpreted as software gate coverage, and we therefore avoid presenting seven redundant columns of "100%" as a broad adversarial security result. In the signed-parent-substitution test, the attacker re-signs the altered manifest so the failure is specifically attributable to ancestry validation, not signature verification. Likewise, signed replay and policy substitution are separately tested. The final two classes (S1–S2) identify **out-of-scope semantic/execution claims**; they are not claimed to have been cryptographically attacked or proven in this prototype.

4.4 RQ3: Which verification predicates are responsible?

We re-run the mutation suite under four progressively stronger checkers. Payload hashing catches two of seven tested provenance faults; checking the signer catches two additional faults; typed lineage and version freshness detect two more; and explicit policy identity detects the seventh. Table 3 displays both *measured coverage for the seven injected metadata mutations* and the broader *taxonomy coverage* under the nine-class definition. Importantly, the latter is a **derived diagnostic denominator**, not a measured estimate of future attacker frequencies. Figure 5 disaggregates the nine classes and makes visible that neither an authentic signature nor a correct lineage demonstrates semantic data quality or actual execution.

Table 3 Coverage of the local fault-injection cases and broader taxonomy. All seven fault types had 100 repeated invocations per dataset. The extended denominator includes the two deliberately undetectable claim classes S1–S2.

Validator	P faults	All classes	All (%)
Digest only	2/7	2/9	22.2
Digest + signature	4/7	4/9	44.4
+ Typed ancestry and version	6/7	6/9	66.7
+ Authorized policy	7/7	7/9	77.8

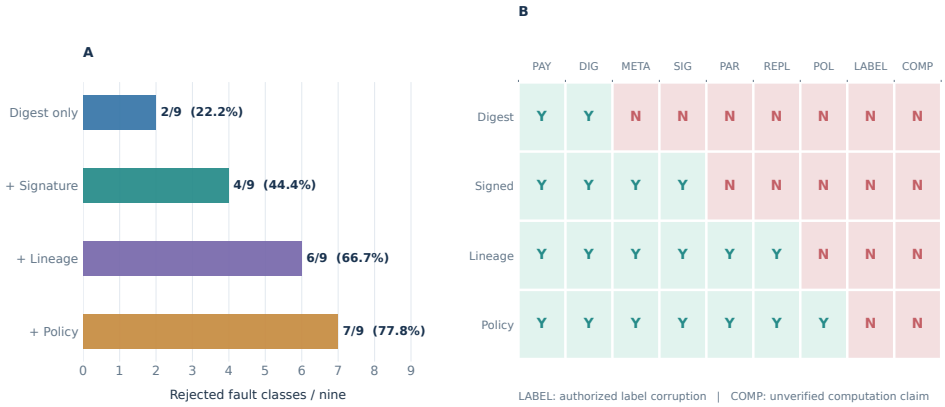


Fig. 5 Predicate coverage by threat class. Left: cumulative fault families rejected by nested metadata-check rules. Right: gate-by-threat matrix; PAY/DIG/META/SIG/PAR/REPL/POL are exercised locally, while LABEL and COMP respectively require semantic assurance and executable computation proofs. These are logical gate outcomes, not estimates of real-world attack prevalence.

4.5 RQ4: How does payload size change local verification cost?

SHA-256 commitment hashing and SHA-256-plus-Ed25519 verification were timed in Python on payloads from 1 KiB to 4 MiB. Each workload was repeated 140 times for sizes below 1 MiB and 65 times for 1 and 4 MiB. Table 4 reports the median and empirical 95th percentile of measured wall-clock microseconds. At small payloads, signature verification dominates; with multi-megabyte checkpoints, reading and hashing the payload dominate. Figure 6 separates the two modes and plots the relative additional verification fraction. These are single-process local microbenchmarks, *not* transaction throughput, EVM gas, blockchain finality, or ZKP runtime.

4.6 RQ5: How does Merkle membership scale?

Binary SHA-256 Merkle trees are constructed for 2^6 to 2^{16} leaves. For each tree, we time 80 membership verifications over randomly sampled indices, after timing construction once. Table 5 records median and empirical 95th-percentile verification time together with proof payload length and one construction time. Verification increases modestly

Table 4 Local CPU operation times (μs). Full local operation is payload SHA-256 plus Ed25519 signature verification, excluding network and ledger.

Payload KiB	Hash med.	Hash p95	Signed med.	Signed p95
1	1.0	1.0	102.1	117.2
32	18.4	18.6	119.6	134.0
256	143.7	164.4	245.8	265.3
1,024	591.5	689.3	697.2	832.6
4,096	2357.4	2615.0	2462.7	2591.4

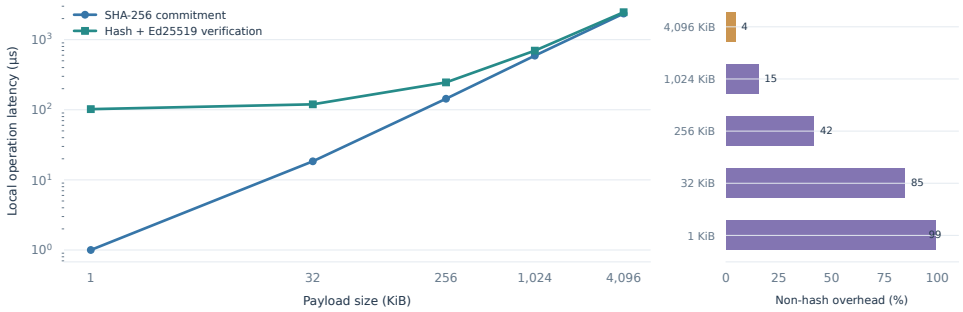


Fig. 6 Local payload-size scaling. Median hashing and signed verification under a controlled one-process benchmark (left), with the approximate fraction of total latency attributable to non-hash work (right). Not on-chain cost.

Table 5 Local Merkle membership. Proof-byte values exclude direction metadata. Building was timed once per tree; results are hardware-sensitive.

Leaves	Depth	Bytes	Med. μs	p95 μs	Build ms
64	6	192	3.99	4.46	0.07
256	8	256	5.34	5.68	0.23
1,024	10	320	6.05	6.29	0.90
4,096	12	384	7.30	7.68	3.50
16,384	14	448	8.32	8.71	14.16
65,536	16	512	10.16	13.28	57.21

with $\log_2 N$, while constructing all leaves scales approximately linearly. Figure 7 separates these quantities into three axes, preventing bytes, microseconds, and milliseconds from being visually conflated.

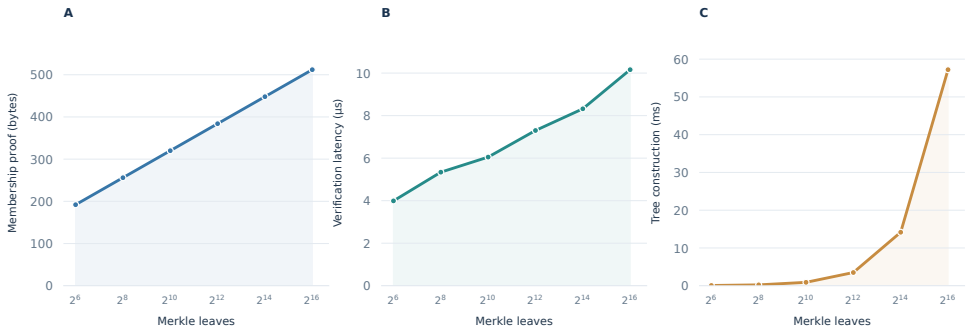


Fig. 7 Six-scale Merkle measurements. (A) Sibling-proof payload size, (B) membership verification time, and (C) end-to-end tree construction time. The measured curves characterize a standalone SHA-256 implementation only.

Table 6 Serialized model payload (KiB) by hidden width; the corresponding SHA-256 digest is exactly 32 B for every entry. Fitted single-hidden-layer MLPs, fixed seed 11.

Dataset	32	48	96	192
Digits	18.8	28.2	56.3	112.6
Breast cancer	8.0	12.0	24.0	48.0
Wine	4.3	6.4	12.8	25.5

4.7 RQ6: Is checkpoint-size decoupling observed?

We train additional MLPs with hidden widths 32, 48, 96, and 192 on the same split of each public dataset (fixed model seed 11). Each fitted parameter array is serialized as contiguous little-endian 64-bit floating-point bytes. Table 6 reports actual resulting checkpoint payload size; the checkpoint digest remains a fixed 32-byte SHA-256 commitment. These values show that the *commitment payload* need not scale with model parameter count, without accounting for contract metadata, signer keys, full proof objects, ledger replication, or off-chain artifact storage. The table reports a measured serialization property, not an end-to-end deployed storage reduction.

4.8 RQ7: What is the finite-population sampling trade-off?

We complement the exact model in Eq. (14) with a *Monte Carlo study* of $N = 100$ committed-step positions. A fixed number b of positions is marked faulty, and k positions are sampled uniformly without replacement; a run detects the fault when its sample intersects the faulty set. For each (b, k) setting, 12,000 Monte Carlo trials estimate the detection frequency, which is compared with the exact hypergeometric probability. Table 7 reports both quantities. This evaluates the statistical sampling mechanism under controlled trace faults; it does not involve on-chain execution, ZKP verification, or independently verified model training. The results are non-perfect when

Table 7 Monte Carlo audit detection (%): detection frequency (exact hypergeometric probability in parentheses), 12,000 sampled trials per setting with $N = 100$. The trials evaluate the sampling rule, independently of blockchain execution.

Faulty b	$k = 5$	$k = 10$	$k = 20$	$k = 30$
1	5.0 (5.0)	9.8 (10.0)	19.8 (20.0)	29.4 (30.0)
3	14.4 (14.4)	26.9 (27.3)	48.9 (49.2)	65.9 (66.1)
5	23.1 (23.0)	41.1 (41.6)	68.3 (68.1)	83.8 (83.9)
10	41.1 (41.6)	66.6 (67.0)	90.5 (90.5)	97.8 (97.7)
15	56.5 (56.4)	81.5 (81.9)	97.4 (97.4)	99.8 (99.7)
20	68.6 (68.1)	90.7 (90.5)	99.3 (99.3)	99.97 (99.97)

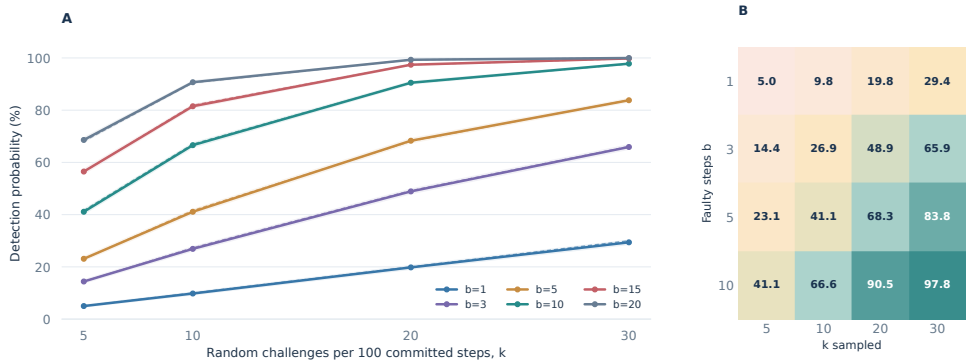


Fig. 8 Monte Carlo detection frequency under randomly placed trace faults (solid) and exact hypergeometric probability (dashed). Shaded bands indicate approximate 95% binomial Monte Carlo uncertainty over 12,000 trials per setting; no blockchain or computation proof is executed.

the adversary corrupts only a small subset, a core reason sampled auditing cannot be reinterpreted as full proof.

4.9 What remains necessary for a systems-level evaluation

Further systems-level validation would require an *executed* Solidity/EVM or Fabric policy contract, real on-chain transaction inclusion/finality, honest and compromised signers, gas and storage accounting over multiple validator counts, and integrated zkVM or zkML proving whose public input binds the very commitment and transition studied here. Model-level studies should further consider larger networks, adaptive data poisoning, compromised custodians, missing off-chain artifacts, time-varying policy versions, finality reorganizations, and comparisons with modern proof-of-training or verified-inference systems. We do not report on-chain gas fees, proof-generation latency, or validator-network throughput because these quantities were not measured.

5 Discussion

What a ledger can and cannot certify.

A ledger provides a durable ordering mechanism and a replicated rule-enforcement surface under its stated finality and validator assumptions. It cannot reveal an undetected poisoned sample, judge the legitimacy of a data license, or make a dishonest but authorized training claim computationally correct. In particular, a fully authenticated lineage can lead to a poor-quality or maliciously trained model. Our claim levels deliberately separate L_0: byte integrity, L_1: authenticated lineage and policy, L_2: relation-valid computation under a sound proof backend, and L_3: semantic and safety assurance requiring additional external evidence.

Relationship to current verifiable machine learning.

The strongest aspect of the proposed formulation is the *binding constraint*: even a cryptographically valid proof is irrelevant if its public statement does not identify the accepted data, transformation, configuration, and parent checkpoint. Systems such as ZKML, zkLLM, and Artemis address practical components of this requirement [10, 11, 14]; the contribution here is an explicit lifecycle-level acceptance relation and policy design. Since cryptographic proof systems and provenance architectures evolve rapidly, no unqualified priority or universal speedup claim is made.

Limitations.

The actual implementation contains no real blockchain execution, no zero-knowledge prover/verifier backend, no trusted random beacon, no robust adversarial learning, and no large-scale network deployment. The seven injected provenance mutations are constructed to exercise the corresponding implemented metadata predicates, so perfect rejection on these narrow cases remains a software-regression result rather than broad security robustness. The evaluation covers seven training seeds and five label-corruption levels, but includes only three small scikit-learn datasets and one MLP family. The audit study uses controlled random fault positions rather than executed on-chain transitions; its agreement with the exact sampling expression validates the sampler rather than a deployed protocol. CPU timing measurements are platform-specific and cannot establish cross-environment speedups. Cryptographic domain-tag conventions in the formal specification are more complete than the compact illustrative Python implementation; migrating to a production contract requires identical canonical encodings and domain separation on both sides. Finally, all published analytical statements are conditional on protocol and cryptographic assumptions; they are not independent end-to-end proofs about a deployed blockchain.

Future work.

Further work should implement a real commit-and-prove backend for selected training/inference operations, compare it fairly against conventional standalone hash anchoring and contemporary zkML, measure ledger gas/finality across validators,

demonstrate policy agility without breaking historical version semantics, and test multi-party model histories with real-world artifact repositories. These tasks are essential for empirical claims of scalable, low-cost, fully verifiable deep learning.

6 Conclusion

We formulated model integrity and provenance as a typed, policy-aware, proof-carrying transition problem, designed a contract-mediated state machine, and separated full computational soundness from bounded post-commitment audit detection. The local prototype demonstrates deterministic digest, signature, ancestry, and policy checks, while 105 public-dataset training runs show that signed provenance can coexist with degraded predictive accuracy under controlled label corruption. Scaling tests quantify local hashing and Merkle cost, and separate Monte Carlo experiments illustrate the limits of incomplete audits. The reported measurements do not demonstrate on-chain performance or fully verified neural training. A production evaluation integrating smart contracts with commitment-bound computation proofs is necessary before the proposed design can substantiate stronger deployment claims.

References

[1] Longpre, S., Mahari, R., Chen, A., et al.: The Data Provenance Initiative: A Large Scale Audit of Dataset Licensing and Attribution in AI (2023). https://arxiv.org/abs/2310.16787

[2] Longpre, S., Mahari, R., Lee, A., *et al.*: Consent in Crisis: The Rapid Decline of the AI Data Commons. In: Advances in Neural Information Processing Systems (2024). https://arxiv.org/abs/2407.14933

[3] Longpre, S., Mahari, R., Obeng-Marnu, N., *et al.*: Position: Data Authenticity, Consent, and Provenance for AI Are All Broken: What Will It Take to Fix Them? In: Proceedings of the 41st International Conference on Machine Learning (2024). https://proceedings.mlr.press/v235/longpre24b.html

[4] Bommasani, R., Klyman, K., Longpre, S., *et al.*: Foundation Model Transparency Reports. In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (2024). https://doi.org/10.1609/aies.v7i1.31628 . https://doi.org/10.1609/aies.v7i1.31628

[5] Bommasani, R., Klyman, K., Longpre, S., et al.: The 2023 Foundation Model Transparency Index. Transactions on Machine Learning Research (2025)

[6] Abdu, S.M., *et al.*: Blockchain-Audited Federated Learning: Securing Data and Model Updates with On-Chain Provenance. IET Software (2025) https://doi.org/10.1049/sfw2/6670439

[7] Abdu, S.M., *et al.*: Trustworthy AI for Secure and Robust Machine Learning Through Blockchain Enabled Data Integrity. Discover Artificial Intelligence (2026)

https://doi.org/10.1007/s44163-026-01443-5

[8] Xu, Z., Zhang, X., Yin, X., Cheng, H.: DePoL: Assuring Training Integrity in Collaborative Learning via Decentralized Verification. Journal of Parallel and Distributed Computing (2025) https://doi.org/10.1016/j.jpdc.2025.105056

[9] Ma, J., Liu, H., Zhang, M., Liu, Z.: VPFL: Enabling Verifiability and Privacy in Federated Learning with Zero-Knowledge Proofs. Knowledge-Based Systems (2024) https://doi.org/10.1016/j.knosys.2024.112115

[10] Chen, B.J., *et al.*: ZKML: An Optimizing System for ML Inference in Zero-Knowledge Proofs. In: Proceedings of EuroSys (2024). https://doi.org/10.1145/3627703.3650088 . https://doi.org/10.1145/3627703.3650088

[11] Sun, H., Li, J., Zhang, H.: zkLLM: Zero Knowledge Proofs for Large Language Models. In: Proceedings of ACM CCS (2024). https://doi.org/10.1145/3658644.3670334 . https://doi.org/10.1145/3658644.3670334

[12] Qu, W., Sun, Y., Liu, X., *et al.*: zkGPT: An Efficient Non-Interactive Zero-Knowledge Proof Framework for LLM Inference. In: Proceedings of USENIX Security Symposium (2025)

[13] Abbaszadeh, K., Katz, J., Pappas, C., Papadopoulos, D.: Zero-Knowledge Proofs of Training for Deep Neural Networks. In: Proceedings of ACM CCS (2024). https://doi.org/10.1145/3658644.3670316 . https://doi.org/10.1145/3658644.3670316

[14] Lycklama, H., Viand, A., Avramov, N., Kuechler, N., Hithnawi, A.: Artemis: Efficient Commit-and-Prove SNARKs for zkML (2024). https://arxiv.org/abs/2409.12055

[15] Garg, S., Goel, A., Jha, S., *et al.*: Experimenting with Zero-Knowledge Proofs of Training. In: Proceedings of ACM CCS (2023)

[16] Tan, G., Gascon, A., Meiklejohn, S., Raykova, M., Wang, X., Luo, N.: Founding Zero-Knowledge Proof of Training on Optimum Vicinity. In: Proceedings of ACM CCS (2025). https://doi.org/10.1145/3719027.3744862 . https://doi.org/10.1145/3719027.3744862

[17] Hao, M., *et al.*: Scalable Zero-Knowledge Proofs for Non-Linear Functions in Machine Learning. In: Proceedings of USENIX Security Symposium (2024)

[18] Lu, T., et al.: An Efficient and Extensible Zero-Knowledge Proof Framework for Neural Networks. Cryptology ePrint Archive (2024)

[19] Ghodsi, Z., Gu, T., Garg, S.: SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud. In: Advances in Neural Information Processing Systems (2017)

[20] Liu, T., Xie, X., Zhang, Y.: zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In: Proceedings of ACM CCS (2021)

[21] Lee, S., et al.: vCNN: Verifiable Convolutional Neural Network Based on zk-SNARKs. IEEE Transactions on Dependable and Secure Computing (2024)

[22] Fan, Y., Xu, B., Zhang, L., et al.: Validating the Integrity of Convolutional Neural Network Predictions Based on Zero-Knowledge Proof. Information Sciences (2023)

[23] Weng, J., et al.: pvCNN: Privacy-Preserving and Verifiable Convolutional Neural Network Testing. IEEE Transactions on Information Forensics and Security (2023)

[24] Maheri, M.M., Haddadi, H., Davidson, A.: TeleSparse: Practical Privacy-Preserving Verification of Deep Neural Networks (2025). https://arxiv.org/abs/2504.19274

[25] Wang, Y.: Zero-Knowledge Proof Based Verifiable Inference of Models (2025). https://arxiv.org/abs/2511.19902

[26] Sun, H., et al.: zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training. Cryptology ePrint Archive (2023)

[27] Fan, Y., Ma, K., Zhang, L., Liu, J., Xiong, N., Yu, S.: VeriCNN: Integrity Verification of Large-Scale CNN Training Process Based on zk-SNARK. Expert Systems with Applications (2024) https://doi.org/10.1016/j.eswa.2024.124531

[28] Liao, G., Wang, T., Zhang, S., et al.: zkLoRA: Fine-Tuning Large Language Models with Verifiable Security via Zero-Knowledge Proofs (2025). https://arxiv.org/abs/2508.21393

[29] Peng, Z., *et al.*: A Survey of Zero-Knowledge Proof Based Verifiable Machine Learning. Artificial Intelligence Review (2026) https://doi.org/10.1007/s10462-026-11557-y

[30] Sheybani, N., et al.: Zero-Knowledge Proof Frameworks: A Systematic Survey (2025). https://arxiv.org/abs/2502.07063

[31] Setty, S.: Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In: Advances in Cryptology - CRYPTO 2020 (2020)

[32] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-Bases for Oecumenical Noninteractive Arguments of Knowledge. Cryptology ePrint Archive (2019)

[33] Setty, S., Thaler, J., Wahby, R.S.: Unlocking the Lookup Singularity with Lasso. In: Proceedings of EUROCRYPT (2024)

[34] Campanelli, M., Faonio, A., Fiore, D., *et al.*: Lookup Arguments: Improvements,

Extensions and Applications to Zero-Knowledge Decision Trees. In: Proceedings of PKC (2024)

[35] Arun, A., Setty, S., Thaler, J.: Jolt: SNARKs for Virtual Machines via Lookups. In: Proceedings of EUROCRYPT (2024). https://www.microsoft.com/en-us/research/publication/jolt-snarks-for-virtual-machines-via-lookups/

[36] Buenz, B., Chen, B.: ProtoStar: Generic Efficient Accumulation/Folding for Special Sound Protocols. In: Advances in Cryptology - ASIACRYPT 2023 (2023). https://eprint.iacr.org/2023/620

[37] Chen, B., Buenz, B., Boneh, D., Zhang, Z.: HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In: Advances in Cryptology - EUROCRYPT 2023 (2023)

[38] Kothapalli, A., Setty, S.: HyperNova: Recursive Arguments for Customizable Constraint Systems. In: Advances in Cryptology - CRYPTO 2024 (2024). https://www.microsoft.com/en-us/research/publication/hypernova-recursive-arguments-for-customizable-constraint-systems/

[39] Xing, Z., et al.: Zero-Knowledge Proof-Based Practical Federated Learning on Blockchain (2023). https://arxiv.org/abs/2304.05590

[40] Xing, Z., *et al.*: Zero-Knowledge Proof-Based Verifiable Decentralized Machine Learning in Communication Network: A Comprehensive Survey. IEEE Communications Surveys and Tutorials (2025) https://doi.org/10.1109/COMST.2025.3561657

[41] Mazzocca, C., Mora, A., Romandini, N., *et al.*: I Trained That! Client-Side Proof of Participation in Federated Learning. Cluster Computing (2025) https://doi.org/10.1007/s10586-025-05668-4

[42] Noh, S., Rhee, K.-H.: Transparent and Accountable Training Data Sharing in Decentralized Machine Learning Systems. Computers, Materials and Continua (2024) https://doi.org/10.32604/cmc.2024.050949

[43] Gu, B., Nawab, F.: zk-Oracle: Trusted Off-Chain Compute and Storage for Decentralized Applications. Distributed and Parallel Databases (2024) https://doi.org/10.1007/s10619-024-07444-6

[44] Zhan, Q., *et al.*: Validating the Integrity for Deep Learning Models Based on Zero-Knowledge Proof and Blockchain. In: Blockchain and Web3 Technology Innovation and Application Exchange Conference (2024)

[45] Sukumaran, S., Korath, A., Menon, G.A.: Tamper-Evident Data and Model Provenance for IoT-Based Machine Learning Using Blockchain and Off-Chain Storage. Information (2026) https://doi.org/10.3390/info17050499

[46] Kersic, V., Karakatic, S., Turkanovic, M.: On-Chain Zero-Knowledge Machine Learning: An Overview and Comparison. Journal of King Saud University Computer and Information Sciences (2024) https://doi.org/10.1016/j.jksuci.2024.102207

[47] Mitchell, M., Wu, S., Zaldivar, A., *et al.*: Model Cards for Model Reporting. In: Proceedings of the Conference on Fairness, Accountability, and Transparency (2019). https://doi.org/10.1145/3287560.3287596 . https://doi.org/10.1145/3287560.3287596

[48] Gebru, T., Morgenstern, J., Vecchione, B., *et al.*: Datasheets for Datasets. Communications of the ACM (2021) https://doi.org/10.1145/3458723