ROBUSTLY CONTESTABLE OWNERSHIP PROOF FROM MODEL MEMORIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Open-weight models can be adapted and redistributed: Alice trains and releases a model, Bob then fine-tunes it but claims he trained it himself. How can Alice provide a publicly verifiable proof that Bob's model is derived from hers? We study this goal as a *contestable ownership proof*: Alice wants to prove that she trained her model and wants to contest any declared lineage that omits her release. Despite progress in model *fingerprinting*, which embeds detectable signals in models, existing methods do not enable Alice to publicly prove that Bob's model was derived from hers. In this paper, we present the first such construction: a generic framework that uses cryptography to bind a model fingerprint to its developer's identity and declared lineage, enabling any third party to verify ownership proofs and lineage challenges. We instantiate the framework with palimpsestic memorization, the tendency of models to memorize later training data more strongly. We implement the resulting protocol as a Python library. To test the robustness of this instantiation, we study adversarial fine-tuning that aims to make the original developer's ownership claim fail verification. We evaluate six models of different sizes from three families: OLMo, Pythia, and CrystalCoder. For each model, we use linear programming to find a fine-tuning direction predicted to reduce the fingerprint signal as much as possible. Along these tested directions, the ownership test stops detecting the signal only after WikiText perplexity rises substantially.

1 INTRODUCTION

Open-weight models allow developers to download weights, adapt them to new tasks, and redistribute the resulting models. However, such openness can also cause disputes over *model lineage*: which released model was derived from which (Franzen, 2024; Global Times, 2024; Reuters, 2025). Suppose Alice trains and releases an open-weight model. Bob may republish it as his own, or fine-tune it and release the result with a lineage that omits Alice's model. How can Alice publicly prove that she trained her model, and that Bob's model derives from hers? In this work, we study how developers can substantiate ownership of their models and challenge lineage declarations that omit them. To this end, we propose a contestable ownership proof (COP) protocol.

Informally, we require that contestable ownership proofs satisfy three properties. First, a developer should be able to produce an accepted ownership proof for her own model and models derived from it, but not for unrelated models. Second, a developer should be able to challenge another developer's declared lineage by providing evidence that the released model derives from hers. The challenge should succeed if and only if that derivation holds and the declared lineage omits her model. Third, every proof should be publicly verifiable, i.e., any third party should be able to verify it from public information alone.

Our approach combines model *fingerprints* with cryptographic *commitments* and a *public append-only ledger* to make ownership claims of open-weight models publicly verifiable and contestable. We instantiate the fingerprint using *palimpsestic memorization* (Kuditipudi et al., 2025): language models tend to memorize examples encountered later in training more strongly, creating a detectable correlation between when examples were seen during training and their loss under the model. To connect this signal to a particular developer, before training, Alice registers a commitment to her data corpus and initial model weights on the public ledger. The protocol then determines her training order from her public identifier, the commitment, and fresh public randomness. Together, these

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

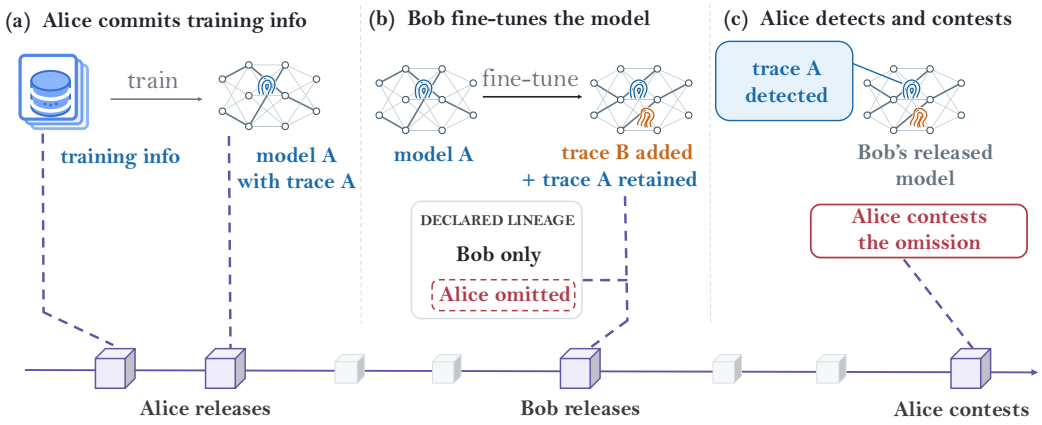


Figure 1: Alice can contest an incomplete declared lineage because her training trace persists after fine-tuning and her release is provably earlier than Bob's. In (a), Alice commits her training information and releases model A. Then, in (b), Bob fine-tunes the model but his declared lineage omits Alice. Using our protocol, in (c), when Alice detects her retained trace in Bob's released model, she can contest the omission. The bottom row represents a public append-only ledger, such as a blockchain, with blocks appended chronologically from left to right. Dashed lines mark what is written to the blockchain. A written record is publicly checkable and irrevocable.

steps fix the claimed training information in advance and tie the prescribed order to Alice's identity, preventing another developer from simply reusing her training order under a different identity. After training, Alice registers a record of her model and its declared lineage on the ledger before releasing the weights, establishing a publicly verifiable chronology of releases.

To substantiate her ownership claim, Alice reveals a publicly selected sample of her committed data, together with cryptographic proofs authenticating the revealed examples. Anyone can verify these proofs, reconstruct the examples' prescribed training positions, and test whether their losses under the released model exhibit the expected statistical dependence. If Bob later releases a model derived from Alice's while omitting hers from its declared lineage, Alice can apply the same procedure to Bob's model. A challenge succeeds when the required ownership proofs verify, Alice's release precedes Bob's, and his declared lineage omits hers. Thus, the memorization signal supplies evidence of derivation, while the cryptographic records bind that evidence to registered claims and establish their chronology. This approach depends on the signal surviving modifications to the model. We therefore evaluate fine-tuning updates explicitly optimized to weaken the signal within a specified attack class, examining the tradeoff between evading detection and preserving model utility.

In summary, our contributions are:

- We propose a generic framework that combines cryptographic tools with a model fingerprint test to obtain a contestable ownership proof.
- We instantiate the framework with a fingerprint based on palimpsestic memorization, implement it as a Python library, and report its storage and runtime cost.
- We analyze how well the palimpsestic memorization signal survives adversarial fine-tuning that is aimed at eliminating the signal while preserving its utility.

2 Related Work

We now discuss a set of works most related to ours.

Post-hoc model-relatedness testing. These tests compare features of two released models' outputs (Cao et al., 2021; Li et al., 2021; Chen et al., 2022; Jin et al., 2024; Lukas et al., 2021; Merkhofer et al., 2023; Tong et al., 2026; Li et al., 2026) and/or their weights and internal representations (Shang et al., 2026; Zhang et al., 2025b; Zeng et al., 2024; Zhu et al., 2025; Yang et al., 2026) to assess

whether they share a training history: for example because one is a copy or fine-tuned version of the other, or because both were derived from a common base model. See also Shao et al. (2025) for a survey. Such tests generally identify relatedness without establishing the direction of derivation. Two works go further: Nikolic et al. (2025, Secs. 2–3.1) test whether a model presented as trained from scratch was fine-tuned from a candidate parent, and Horwitz et al. (2025) use statistics of the weights to infer which model was derived from which. However, none of these works provides a protocol through which a developer can publicly challenge a declared lineage that omits her model.

Training-data testing. Given a model and a dataset, these tests detect whether the model was trained on that dataset (Maini et al., 2021; 2024); Kuditipudi et al. (2025) also uses training order to link a model to a particular training run. They rely on the observation that a model fits its training data better than data it has never seen (Shokri et al., 2017; Yeom et al., 2018). A training data test match is, however, not an ownership proof: independent models trained on similar data can also match (Szyller et al., 2023; Zhang et al., 2025a), and an adversary who inspects a released model can fabricate a data match for a model it did not train (Liu et al., 2024).

Secret-key model fingerprints. These schemes, also known as model watermarking, embed a fingerprint into the model before release, for example in its weights (Uchida et al., 2017), in dedicated architectural components (Fan et al., 2019; Liu et al., 2023), or in its responses to selected inputs (Adi et al., 2018; Xu et al., 2024; Nasery et al., 2025; Chen et al., 2026b; Ding et al., 2026; Chen et al., 2026a; Zhang et al., 2026; Xiong et al., 2026). A developer can then use the presence of that fingerprint in another model as evidence that the other model was copied or derived from hers. In many schemes, verification requires disclosing secret information, which can facilitate removal of the fingerprint and undermine security after public verification. Zero-knowledge approaches avoid this disclosure (Adi et al., 2018, App. A.1) (Sheybani et al., 2023), but fingerprint verification alone does not resolve disputes over declared lineage. A second developer may embed her own fingerprint into a model derived from the first developer's model, so both fingerprints may be present. The validity of both fingerprints alone does not, by itself, establish which model was derived from which.

Theoretic model ownership and computational limits. These works ask whether an owner can embed an ownership proof that no efficient (probabilistic polynomial time) adversary can remove (Goldwasser et al., 2022; Canetti et al., 2026; Cohen et al., 2018; Zhang et al., 2024). These works formalize model ownership and prove impossibility results for some function families (Canetti et al., 2026, Secs. 2–3)(Cohen et al., 2018, Sec. 9). Under cryptographic assumptions, Canetti et al. (2026) also construct black-box ownership proofs for suitable families of classification functions. Our results complement this theoretical characterization by studying contestable ownership for open-weight models. We develop a protocol for publicly verifiable lineage challenges and evaluate its robustness against adversaries with access to model weights and gradients, restricting their updates to an explicitly defined class under a stated resource budget.

Proof of learning. These schemes allow a developer to provide evidence that a model was produced by a reported training process. The developer supplies training data and intermediate model checkpoints, and a verifier re-executes selected portions of training to check consistency with those records (Jia et al., 2021; Choi et al., 2023; Chang et al., 2025; Garg et al., 2023). These works aim to make fabricating convincing training records as expensive as performing the training itself, although substantially cheaper forgeries have been demonstrated (Zhang et al., 2022; Fang et al., 2023; Abbaszadeh et al., 2024). Verifying a reported training process, however, does not by itself establish where its starting model came from. Bob could privately fine-tune Alice's model, then use the resulting checkpoint as the starting point for a training process for which he provides a valid proof. The proof would substantiate that subsequent training without revealing the earlier derivation from Alice's model. Thus, proof of learning alone does not enable Alice to contest a lineage declaration that omits her model.

In summary, none of the approaches surveyed above provides the combination of lineage evidence and publicly verifiable challenges needed for lineage contestability.

3 Problem Formulation and Security Goals

We consider the following scenario. Suppose Alice releases model θ_A at time t_A; Bob releases θ_B at time $t_B > t_A$. Each party is required to declare its model's *lineage* by listing every model used to produce it, as well as every model used to produce those models, recursively. The declaration is empty if the model was trained from scratch. A developer who suspects that another model was derived from hers may challenge that model's declared lineage if her model is omitted. Both parties may behave maliciously; in particular, they may omit their actual base model from their declaration to conceal their model's origin or incorrectly claim that the other party's model was derived from theirs.

Contestable ownership proofs. To let parties securely establish the lineage of their models, we introduce *contestable ownership proof* systems. A contestable ownership proof system consists of six algorithms:

- $\mathsf{Setup}(1^\lambda)$ takes a security parameter λ and outputs public parameters pp used implicitly by the remaining algorithms.
- $\mathsf{Train}(\mathsf{id}, X, \theta_0, \mathcal{P})$ takes the developer's public identifier id, a data corpus X, an initial model θ_0, and a predecessor declaration $\mathcal{P}$. It returns a public ownership claim c, a trained model θ_c, and private information st_c retained for proof generation. The initial model can be either a fresh initialization or a pretrained model used as the starting point for fine-tuning.
- $\mathsf{Prove}(c, \theta, \mathsf{st}_c)$ produces an ownership proof Π for the ownership claim c on the model θ.
- $\mathsf{Verify}(c, \theta, \Pi)$ uses public information to accept or reject the ownership proof.
- $\mathsf{Challenge}(c, c', \mathsf{st}_c)$ takes the challenger's ownership claim c, the claim under dispute c', and the challenger's private information. It produces a challenge Π_{ch}.
- $\mathsf{Judge}(c, c', \Pi_{\mathrm{ch}})$ uses public information to accept or reject the challenge.

Security requirements. We describe four intended security properties informally below. Let $\mathcal{A}$ denote the class of model transformations under consideration, which may comprise multiple successive modification steps.

- **Completeness.** An honest developer's ownership proof for the model she releases will be accepted.
- **Ownership soundness.** A malicious Alice cannot produce an accepted ownership proof for Bob's model θ_B when Alice's model is not part of Bob's lineage.
- **Contestability completeness.** If Bob derives his model from Alice's through a modification in $\mathcal{A}$ and publishes a valid ownership proof while omitting Alice's claim from his declared lineage, Alice can produce an accepted challenge. This property holds even when Bob knows the protocol and deliberately attempts to suppress evidence of derivation.
- **Contestability soundness.** A malicious developer cannot win a false lineage challenge. In particular, a challenge is rejected if the model being challenged does not derive from the challenger's model, or the declared lineage already acknowledges the challenger's claim.

Scope and utility tradeoff. These properties describe the intended security goals of a contestable ownership proof framework, with errors quantified by specified probabilities. While establishing theoretical bounds on these errors for our scheme is an interesting direction for future work, in Section 5 we empirically investigate a key component of its security: the robustness of the statistical evidence underlying ownership claims against adversarial fine-tuning.

We note that there is an inherent trade-off between erasing the evidence of derivation and preserving the model's utility. An adversary willing to destroy a model's utility can erase this evidence, for example by replacing its weights with a fresh initialization. A developer seeking to redistribute a useful model, however, has reason to preserve its capabilities. In Section 5, we therefore evaluate a specific class of adversarial updates and examine the tradeoff between evading detection and retaining model utility.

4 OUR CONSTRUCTION

Our construction binds a statistical model fingerprint to a developer's identity and declared lineage, enabling public verification of ownership proofs and lineage challenges. We defer the details of our generic framework to the Appendix B.1.

Below, we describe a concrete instantiation of our framework with a fingerprint based on LLM memorization: training leaves a statistical signal reflecting *which examples* were used and in *which order*, with examples encountered later tending to have lower loss (Jagielski et al., 2023; Lesci et al., 2024; Chang et al., 2024; Huang et al., 2024; Kuditipudi et al., 2025). Appendix A illustrates both effects. Throughout this paper, $L(\theta; x)$ denotes the negative log-likelihood of example x under model θ. The corrected loss is $L(\theta; x) - L(\theta_{\mathrm{ref}}; x)$ for a fixed reference model θ_{ref}. The fingerprint measures the correlation between a model's per-example corrected losses and the examples' positions in a fixed ordered training schedule [1]. A model trained on that schedule can exhibit detectable loss-order dependence, and this signal can remain detectable in models derived from it. We assess this dependence using a permutation test, comparing the observed correlation with correlations obtained by randomly permuting all position labels, including those assigned to unused examples, while keeping the model and its losses fixed.

We note that using such fingerprint alone is not sufficient to achieve a contestable ownership protocol. Once a model is released, an attacker can fabricate an ordered dataset that exhibits the same loss–order correlation as the true owner's training data, even if its examples were never used to train the model. From this statistical evidence alone, a third party cannot distinguish the true owner from an attacker presenting the fabricated dataset. Our cryptographic techniques prevent such post-hoc selection of the claimed training data and order. Moreover, they bind the statistical evidence to a developer's ownership claim and enable her to challenge later releases derived from her model that omit it from their declared lineage.

Our construction relies on *cryptographic commitments*, which allow a party to fix a message while keeping it secret and reveal it later. They provide two security properties: *hiding*, which conceals the committed message, and *binding*, which prevents the committer from opening the same commitment to two distinct messages. Finally, we rely on a public *random beacon* to obtain fresh public randomness, and a public *append-only ledger*, such as a blockchain, to maintain a publicly verifiable record of commitments and model releases. This record allows anyone to verify that a developer registered a commitment before releasing a model and to determine the order in which models were released. See section B.1 for formal definitions.

At a high level, the protocol consists of the following key steps.

1. Prior to training, Alice, who intends to make the model open source, publishes her public identifier id_A and the cryptographic commitment com to the full data corpus in a fixed canonical order, a training fraction α, and the initial model weights. The fraction specifies that $\lfloor \alpha N \rfloor$ examples will be used for training.
2. After her commitment is finalized, Alice obtains fresh public randomness from the beacon and combines it with her public identifier and corpus commitment to derive the training schedule. This schedule specifies which corpus examples are used and assigns each used example a training order.
3. Alice trains her model θ_A following the training schedule[2].
4. Alice records a hash of θ_A on the ledger under her public identifier, linking it to her registered commitment and lineage declaration. Once this record is finalized, she makes the model weights publicly available. The record fixes the particular model associated with her claim and allows anyone to check when its release was registered.
5. Alice publishes an ownership proof for her own released model. To produce such proof for a target model θ, Alice reveals a publicly determined sample of examples from her committed corpus. This sample, called the *audit panel*, is selected using fresh public randomness generated via the same public random beacon after the target model has been fixed. For every revealed example, she supplies a cryptographic opening proof showing that it is exactly

[1] Assigning position zero to unused examples.

[2] Each run may use only part of the committed corpus.

the entry at the requested index of the committed training data. These proofs authenticate the disclosed examples without requiring disclosure of the entire corpus.

6. Any third party can verify the ledger records and commitment openings, reconstruct the prescribed training positions of the revealed examples, and apply the statistical test to their corrected losses under θ. The proof is accepted if the cryptographic checks pass and the test detects significant dependence between the corrected losses and the prescribed training positions.
7. Suppose Bob subsequently releases a model θ_B with his ownership proof and a declared lineage that omits Alice's model. Alice can challenge this declaration by producing an ownership proof for θ_B under her original commitment. The public judge checks Alice's proof on θ_A, Bob's proof on θ_B, and Alice's proof on θ_B. It also checks that Alice's release was registered before Bob's and that Bob's declared lineage does not acknowledge Alice's model. If all these checks pass, the judge accepts the challenge.

5 Robustness Analysis and Experiment

Kuditipudi et al. (2025) show that the palimpsestic memorization signal survives benign fine-tuning and model souping, i.e., averaging the weights of multiple finetuned models. However, they do not consider an adversary who knows how the palimpsestic memorization test works and deliberately fine-tunes the model to erase the correlation between example loss and training order. In our setting, we analyze an adversary who knows our ownership verification procedure and uses a limited fine-tuning data budget to weaken the statistical evidence supporting an ownership claim while preserving utility (see Appendix A.1 for a formal definition). Motivated by prior work on certified robustness by Bansal et al. (2022); Raghunathan et al. (2018); Wong & Kolter (2018); Cohen et al. (2019); Dathathri et al. (2020), we derive the exact maximum first-order weakening of the ownership statistic within the specified frozen-gradient update class. We then replay the resulting direction and measure its effect on model utility. Our key observation is that if the test statistic F – the Pearson correlation in our case – is differentiable, its gradient gives the direction of steepest signal removal. In the resulting linearized regime, finding the adversary's best update reduces to a linear program under the data-budget constraint.

Adversarial fine-tuning robustness The adversary starts from θ_0 and takes a sequence of fine-tuning steps. At each step j, having reached model θ_j, it picks an example from its attack set $\mathcal{C}$ of $|\mathcal{C}|$ candidate examples - data it has and can compute gradients on. The attack set is usually distinct from the verifier's audit panel, which is sampled independently only after the modified model has been released. We nonetheless give the adversary the exact audit panel in advance, making this threat model substantially stronger than a practical removal attack. Across these steps the adversary assigns each a signed step size $\eta_j \in \mathbb{R}$: a positive step reinforces that example (learning), a negative step works against it (unlearning). Let $F(\theta)$ be the resulting test statistic; our experiments use the Pearson correlation, across audit-panel sequences, between the registered training-stream coordinate—set to zero for occurrences beyond the checkpoint—and the corrected loss, defined as target-model total NLL minus fixed-reference total NLL. The adversary's goal is to make the verifier's test non-significant: p-value > 0.01, the significance level used throughout our experiments.

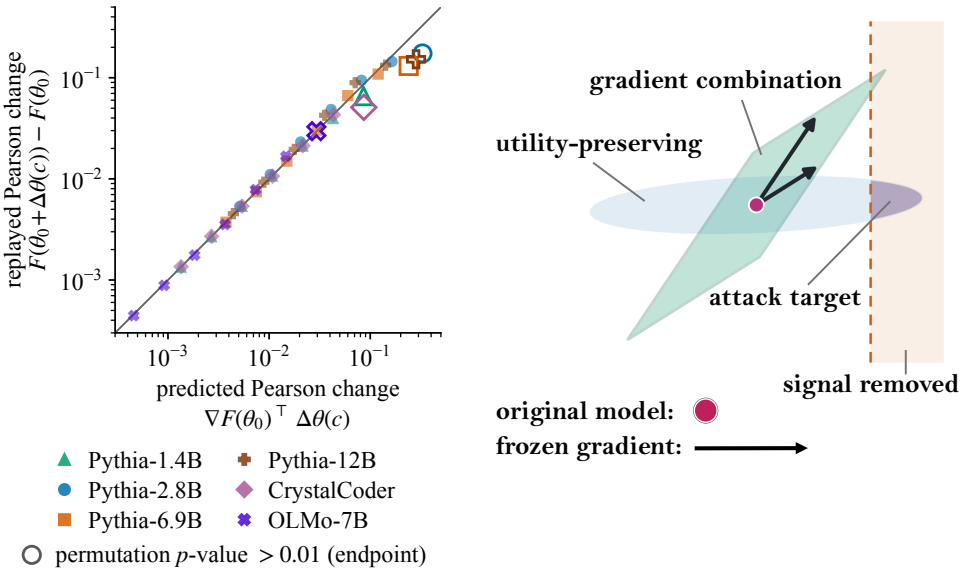
We study this adversary in the *frozen gradient model*. In ordinary fine-tuning, each step follows the gradient of the chosen example at the model as it currently stands. The frozen gradient model keeps the same steps and the same examples, but takes each example's gradient at the starting model and reuses it throughout. Previous work supports this simplification: Under suitable conditions, a wide network trained by gradient descent stays close to its linearization around its starting point (Jacot et al., 2018; Lee et al., 2019; Chizat et al., 2019); for language models, gradients taken at the pretrained weights predict what fine-tuning will do - a kernel built from them recovers most of the accuracy of full fine-tuning (Malladi et al., 2023), and a first-order expansion at the pretrained weights predicts the fine-tuning loss to within about one percent (Li et al., 2024). See Appendix A.1 for further evidence.

The frozen gradient model makes the adversary tractable. Write $g_i(\theta) = \nabla_\theta \ell(\theta; x_i)$ for the gradient of example x_i at model θ. Every step that touches example x_i now uses the same vector $g_i(\theta_0)$, so all the steps on that example add up to a single coefficient $c_i := \sum_{j:\, i_j = i} \eta_j$, and the whole sequence of

steps becomes one linear combination over the attack set:

$$\Delta\theta(\boldsymbol{c}) = -\sum_{x_i \in \mathcal{C}} c_i g_i(\theta_0), \qquad |c_i| \le \bar{\eta}. \tag{1}$$

This lets us bound the adversary's strategy by the $|\mathcal{C}|$ examples it holds rather than its individual step-by-step choices, so that we do not have to brute-force every possible sequence of training steps — a space that is hard to characterize with a reasonable threat model and impractical to bound directly. Figure 2b illustrates the geometry. We constrain the coefficients by $|c_i| \le \bar{\eta}$ and assume that, and assume that the change to the statistical can be approximated by first-order Taylor:



(a) (b)

Figure 2: (a) Predicted versus replayed changes in the Pearson statistic along the tested optimal frozen-gradient updates for the same six model–configuration cells shown in Figure 3a. Predicted changes closely track replayed changes across the tested scales, with larger deviations at the largest replay scale, providing empirical support for the first-order approximation on these updates. (b) The schematic geometry this approximation describes. Each point represents a model's parameters. The green region shows updates reachable by combining frozen gradients under the attack budget. Along the tested signal-removal directions, the first checkpoints that evade detection have substantially higher WikiText perplexity (Figure 3a).

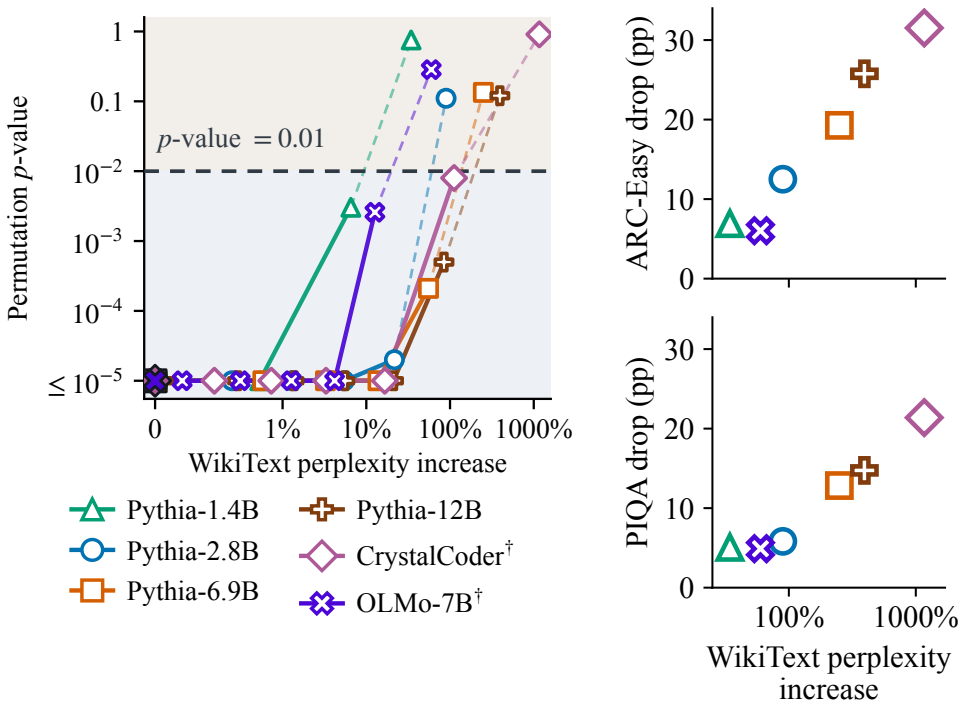
> **Linear approximation.** The change in the statistic, $F(\theta_0 + \Delta\theta(\boldsymbol{c})) - F(\theta_0)$, is approximated by its first-order Taylor expansion $\nabla F(\theta_0)^\top \Delta\theta(\boldsymbol{c})$, with error at most δ_F over the declared update region.

Because $\Delta\theta$ is linear in $\boldsymbol{c}$, the adversary's best strategy - finding the feasible c that moves F furthest toward non-significance under the constraints of Equation 1 reduces to a linear program.

Multiple models and data corpus. We evaluate order-loss correlation signal removal versus utility drop, under the optimal frozen-gradient update of Section 5, on Pythia at 1.4B, 2.8B, 6.9B, and 12B, on CrystalCoder, and on OLMo-7B. Each target checkpoint uses approximately the first half of its corpus in training ($\alpha \approx 1/2$); the remaining occurrences are unused.[3] The attack set and audit panel are both drawn from this corpus: Pythia configurations use 10K sequences for each, while OLMo-7B and CrystalCoder use a 100K attack set and a 100K audit panel, since their noisier signal makes the Pearson test unstable at 10K sequences test panel.

[3]The available Pythia family runs use step 71,000 of 143,000 (49.65%); CrystalCoder uses phase-1 step 40,500 of 79,721 (50.80%); and OLMo-7B uses 466,560,000 of 934,005,600 eligible presentations (49.95%). Each reported configuration records its actual prefix length.

For every Figure 3a cell, scoring uses the first 128 next-token positions of each sampled sequence (129 input tokens). Token losses are computed in binary32 and reduced to a sequence total in binary64; the corrected response is the target total minus the fixed-reference total. Candidate gradients use the target model's mean next-token loss over the same 128 positions. The coordinate convention is fixed before scoring and reused at every replay endpoint: seen occurrences use one-based presentation positions, and future occurrences are zero.



(a) (b)

Figure 3: (a) Permutation p-values of the ownership test against WikiText perplexity increase for checkpoints replayed along each model's optimal signal-removal direction (filled markers: unmodified models; a curve turns dashed where it crosses $p = 0.01$; values at or below the permutation floor 10^{-5} are drawn at the floor; the horizontal axis is logarithmic and increases of at most 0.03% are drawn at 0). †: 100K attack set and 100K audit panel; other models 10K and 10K. (b) ARC-Easy (top) and PIQA (bottom) accuracy drop, in percentage points, at the six endpoints of (a), the first checkpoint at which the test no longer detects the signal, against the WikiText perplexity increase at that checkpoint (log scale); same models and markers as (a).

We use WikiText perplexity as the primary proxy for model utility, together with ARC-Easy and PIQA accuracy (Figure 3b). Our robustness conclusions therefore concern utility as measured by these declared metrics; they do not establish preservation of every downstream use. These benchmarks are nonlinear functions of the update that a first-order gradient approximation cannot capture, so they do not enter the linear program; only the linear approximation of F does. We compute the combination of candidate gradients that optimally moves F toward the statistical non-significance boundary, p-value > 0.01. This optimal combination defines the optimal signal-removal direction, $\Delta\theta(\boldsymbol{c}^*)$ (Equation 1). We then validate the linear approximation directly by replaying the attack with the optimal combination for each model, comparing the predicted signal removal against the signal removal actually measured after replay; the predicted changes closely track replayed changes across the tested scales, with larger deviations at the largest replay scale (Figure 2a). We evaluate utility benchmark on each replayed checkpoint along the optimal direction at increasing scales, and report the resulting signal–utility tradeoff.

Results. Moving weights along the optimal signal-removal direction in our frozen-gradient threat model, p-value stays significant until it has cost substantial utility (Figure 3a). All unmodified models

are strongly detected ($p \leq 10^{-5}$). At the last checkpoint that is still significant, the attack has already raised WikiText perplexity by 6.5% (Pythia-1.4B) to 112% (CrystalCoder); the first checkpoint that is no longer detected costs 34% to $1{,}166\%$, and more than 100% for Pythia-6.9B, Pythia-12B, and CrystalCoder. The same endpoints cost downstream accuracy: ARC-Easy falls by 6.0 to 31.5 points and PIQA by 4.9 to 21.4, and the drop broadly tracks the perplexity increase (Figure 3b).

The linear program identifies the strongest first-order signal-removal update among all admissible combinations of attack-set gradients under the stated coefficient cap. Replaying this update connects the optimization to measured ownership-test outcomes: across all six models, the first tested checkpoint at which the permutation test no longer detects the signal incurs substantial WikiText perplexity increases and downstream accuracy losses. Figure 2a assesses prediction–replay agreement, while Figure 3a reports the resulting signal–utility tradeoff along these optimized attack directions. The prediction–replay agreement in Figure 2a measures how faithfully the frozen model describes the replayed updates. Figure 3a reports the complete-row, two-sided, plus-one permutation p-value at each replay scale. The removal endpoint is the first replay scale with $p_{\mathrm{perm}} > 0.01$.

Figure 5 compares four Pythia model sizes with an attack set identical to the 10K audit panel. At the first measured non-significant replay checkpoint, WikiText perplexity increases by 35.1%, 89.4%, 250.6%, and 393.9% for Pythia-1.4B, 2.8B, 6.9B, and 12B, respectively.

5.1 Protocol Implementation and Efficiency

Lineage, our PyTorch extension package, implements corpus commitments, audit sampling, proof generation, and statistical verification, and interface to public ledger. Table 1 summarizes measured component costs and estimated full-corpus costs for a 10K-sequence audit panel. End-to-end runtime has not yet been measured.

Table 1: Protocol costs for a 10K-sequence audit panel on each model's training corpus. Corpus commitment costs are incurred once per corpus and estimated from measured reading, writing, and hashing rates. Proof generation and verification costs are incurred once per proof. Proof opening and serialization costs are measured directly. Verification is dominated by scoring the target and reference models on one H100. For Pythia-12B, CrystalCoder-7B, and OLMo-7B, target-model scoring is measured, while reference-model scoring uses a symmetric cost estimate.

		Owner, once per corpus		Per proof		
Corpus	Tokens	PreCommit	Extra storage	Size	Prove	VerifyProof
Pythia 1.4B–12B	300B	52 min	13 GiB	48 MiB	1.9 s	2.8–26 min
CrystalCoder-7B	345B	60 min	15 GiB	48 MiB	1.9 s	14 min
OLMo 7B	4T	21 h	87 GiB	166 MiB	2.2 s	14 min

6 Conclusion

We propose contestable ownership proofs, which let the developer of an open-weight model prove that she trained it and contest any declared lineage that omits it. Our construction combines model fingerprints with cryptographic commitments registered before training and a public record of model releases, binding ownership evidence to developers and establishing release chronology. Both ownership proofs and lineage challenges are publicly verifiable.

We instantiate the framework with palimpsestic memorization, which leaves a statistical trace of training order in a model's losses. Because a lineage dispute invites tampering, we ask whether a developer who knows the test can fine-tune the trace away while keeping the model useful. On six open-weight models, along the tested signal-optimal frozen-gradient directions, the first measured non-significant endpoints checkpoints show substantially higher WikiText perplexity and lower ARC-Easy and PIQA accuracy.

REFERENCES

Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024. doi: 10.1145/3658644.3670316.

Alessandro Achille, Aditya Golatkar, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. Lqf: Linear quadratic fine-tuning, 2020.

Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

Zahra Rahimi Afzal, Tara Esmaeilbeig, Mojtaba Soltanalian, and Mesrob I. Ohannessian. Linearization explains fine-tuning in large language models, 2026.

Arpit Bansal, Ping-Yeh Chiang, Michael J Curry, Rajiv Jain, Curtis Wigington, Varun Manjunatha, John P Dickerson, and Tom Goldstein. Certified neural network watermarks with randomized smoothing. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

Ran Canetti, Shafi Goldwasser, and Or Zamir. Proofs of ownership for machine learning models. *arXiv preprint arXiv:2606.30423*, 2026. doi: 10.48550/arXiv.2606.30423.

Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. IPGuard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021. doi: 10.1145/3433210.3437526.

Hoyeon Chang, Jinho Park, Seonghyeon Ye, Sohee Yang, Youngkyung Seo, Du-Seong Chang, and Minjoon Seo. How do large language models acquire factual knowledge during pretraining? In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-1939.

Yijia Chang, Hanrui Jiang, Chao Lin, Xinyi Huang, and Jian Weng. Towards understanding and enhancing security of Proof-of-Training for DNN model ownership verification. In *34th USENIX Security Symposium (USENIX Security 25)*, 2025.

Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. Copy, right? a testing framework for copyright protection of deep learning models. In *2022 IEEE Symposium on Security and Privacy*, 2022. doi: 10.1109/SP46214.2022.9833747.

Sixu Chen, Xiang Chen, Hongyao Yu, Jiaxin Hong, Hao Fang, Shuoyang Sun, Bin Chen, and Shu-Tao Xia. Prompt2fingerprint: Plug-and-play llm fingerprinting via text-to-weight generation, 2026a.

Yiwei Chen, Bingqi Shang, and Sijia Liu. Who built this model? tracing llm lineage via spectral fingerprints in weight space, 2026b.

Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems*, 2019.

Dami Choi, Yonadav Shavit, and David K Duvenaud. Tools for verifying neural models' training data. In *Advances in Neural Information Processing Systems*, 2023. doi: 10.52202/075280-0056.

Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017. doi: 10.1145/3133956.3134092.

Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. *SIAM Journal on Computing*, 2018. doi: 10.1137/18M1164834.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy R Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy S Liang, and Pushmeet Kohli. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. In *Advances in Neural Information Processing Systems*, 2020.

Zikang Ding, Junhao Li, Suling Wu, Junchi Yao, Hongbo Liu, and Lijie Hu. Functional subspace watermarking for large language models, 2026.

Lixin Fan, Kam Woh Ng, and Chee Seng Chan. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. In *Advances in Neural Information Processing Systems*, 2019.

Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning is currently more broken than you think. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, 2023. doi: 10.1109/EuroSP57164.2023.00052.

Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M. Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel, 2020.

Carl Franzen. New open source AI leader Reflection 70B's performance questioned, accused of 'fraud'. VentureBeat, September 2024.

Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023. doi: 10.1145/3576915.3623202.

Global Times. Stanford AI project team apologizes for plagiarizing Chinese model. Global Times, June 2024.

Shafi Goldwasser, Michael P. Kim, Vinod Vaikuntanathan, and Or Zamir. Planting undetectable backdoors in machine learning models. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, 2022. doi: 10.1109/FOCS54457.2022.00092.

Eliahu Horwitz, Asaf Shul, and Yedid Hoshen. Unsupervised model tree heritage recovery. In *International Conference on Learning Representations*, 2025.

Jing Huang, Diyi Yang, and Christopher Potts. Demystifying verbatim memorization in large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024. doi: 10.18653/v1/2024.emnlp-main.598.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

Matthew Jagielski, Om Thakkar, Florian Tramèr, Daphne Ippolito, Katherine Lee, Nicholas Carlini, Eric Wallace, Shuang Song, Abhradeep Guha Thakurta, Nicolas Papernot, and Chiyuan Zhang. Measuring forgetting of memorized training examples. In *International Conference on Learning Representations*, 2023.

Uijeong Jang, Jason D. Lee, and Ernest K. Ryu. Lora training in the ntk regime has no spurious local minima, 2024.

Hengrui Jia, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy*, 2021. doi: 10.1109/SP40001.2021.00106.

Heng Jin, Chaoyu Zhang, Shanghao Shi, Wenjing Lou, and Y. Thomas Hou. ProFLingo: A fingerprinting-based intellectual property protection scheme for large language models. In *2024 IEEE Conference on Communications and Network Security*, 2024. doi: 10.1109/CNS62487.2024.10735575.

Rohith Kuditipudi, Jing Huang, Sally Zhu, Diyi Yang, Christopher Potts, and Percy Liang. Blackbox model provenance via palimpsestic membership inference. In *Advances in Neural Information Processing Systems*, 2025. doi: 10.52202/085713-5421.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*, 2019.

Pietro Lesci, Clara Meister, Thomas Hofmann, Andreas Vlachos, and Tiago Pimentel. Causal estimation of memorisation profiles. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. doi: 10.18653/v1/2024.acl-long.834.

Dongyue Li, Ziniu Zhang, Lu Wang, and Hongyang R. Zhang. Scalable fine-tuning from multiple data sources: A first-order approximation approach. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024. doi: 10.18653/v1/2024.findings-emnlp.321.

Siquan Li, Yao Tong, Haonan Wang, and Tianyang Hu. Transformers are born biased: Structural inductive biases at random initialization and their practical consequences. *arXiv preprint arXiv:2602.05927*, 2026. doi: 10.48550/arXiv.2602.05927.

Yuanchun Li, Ziqi Zhang, Bingyan Liu, Ziyue Yang, and Yunxin Liu. ModelDiff: Testing-based DNN similarity comparison for model reuse detection. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021. doi: 10.1145/3460319.3464816.

Hanwen Liu, Zhenyu Weng, Yuesheng Zhu, and Yadong Mu. Trapdoor normalization with irreversible ownership verification. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Jian Liu, Rui Zhang, Sebastian Szyller, Kui Ren, and N. Asokan. False claims against model ownership resolution. In *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep neural network fingerprinting by conferrable adversarial examples. In *International Conference on Learning Representations*, 2021.

Pratyush Maini, Mohammad Yaghini, and Nicolas Papernot. Dataset inference: Ownership resolution in machine learning. In *International Conference on Learning Representations*, 2021.

Pratyush Maini, Hengrui Jia, Nicolas Papernot, and Adam Dziedzic. LLM dataset inference: Did you train on my dataset? In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-3941.

Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. A kernel-based view of language model fine-tuning. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Elizabeth Merkhofer, Deepesh Chaudhari, Hyrum S. Anderson, Keith Manville, Lily Wong, and João Gante. Machine learning model attribution challenge. *arXiv preprint arXiv:2302.06716*, 2023. doi: 10.48550/arXiv.2302.06716.

Anshul Nasery, Jonathan Hayase, Creston Brooks, Peiyao Sheng, Himanshu Tyagi, Pramod Viswanath, and Sewoong Oh. Scalable fingerprinting of large language models. In *Advances in Neural Information Processing Systems*, 2025. doi: 10.52202/085713-4170.

Ivica Nikolic, Teodora Baluta, and Prateek Saxena. Model provenance testing for large language models. In *Advances in Neural Information Processing Systems*, 2025. doi: 10.52202/085713-1146.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models, 2023.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

Reuters. Huawei's AI lab denies that one of its Pangu models copied Alibaba's Qwen. Reuters, July 2025.

Zhuoyi Shang, Jiasen Li, Pengzhen Chen, Yanwei Liu, Xiaoyan Gu, and Weiping Wang. Attesting model lineage by consisted knowledge evolution with fine-tuning trajectory. In *35th USENIX Security Symposium (USENIX Security 26)*, 2026.

Shuo Shao, Yiming Li, Yu He, Hongwei Yao, Wenyuan Yang, Dacheng Tao, and Zhan Qin. SoK: Large language model copyright auditing via fingerprinting. *arXiv preprint arXiv:2508.19843*, 2025. doi: 10.48550/arXiv.2508.19843.

Nojan Sheybani, Zahra Ghodsi, Ritvik Kapila, and Farinaz Koushanfar. ZKROWNN: Zero knowledge right of ownership for neural networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023. doi: 10.1109/DAC56929.2023.10247798.

Elaine Shi. Foundations of distributed consensus and blockchains. `https://elaineshi.com/docs/blockchain-book.pdf`, 2026.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, 2017. doi: 10.1109/SP.2017.41.

Sebastian Szyller, Rui Zhang, Jian Liu, and N Asokan. On the robustness of dataset inference. *Transactions on Machine Learning Research*, 2023.

Yao Tong, Haonan Wang, Siquan Li, Kenji Kawaguchi, and Tianyang Hu. SeedPrints: Fingerprints can even tell which seed your large language model was trained from. In *International Conference on Learning Representations*, 2026.

Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, 2017. doi: 10.1145/3078971.3078974.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Zixun Xiong, Gaoyi Wu, Qingyang Yu, Mingyu Derek Ma, Lingfeng Yao, Miao Pan, Xiaojiang Du, and Hao Wang. iseal: Encrypted fingerprinting for reliable llm ownership verification, 2026. doi: https://doi.org/10.1609/aaai.v40i42.40909.

Jiashu Xu, Fei Wang, Mingyu Ma, Pang Wei Koh, Chaowei Xiao, and Muhao Chen. Instructional fingerprinting of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024. doi: 10.18653/v1/2024.naacl-long.180.

Guang Yang, Ziye Geng, Yihang Chen, and Changqing Luo. Fingerprinting deep neural networks for ownership protection: An analytical approach, 2026.

Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, 2018. doi: 10.1109/CSF.2018.00027.

Boyi Zeng, Lizheng Wang, Yuncong Hu, Yi Xu, Chenghu Zhou, Xinbing Wang, Yu Yu, and Zhouhan Lin. HuRef: HUman-REadable fingerprint for large language models. In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-4013.

Bingxue Zhang, Xiaofeng Xu, and Feida Zhu. Lineagemark: Multi-user white-box watermarking for contribution tracing in model derivation chains, 2026.

Hanlin Zhang, Benjamin L. Edelman, Danilo Francati, Daniele Venturi, Giuseppe Ateniese, and Boaz Barak. Watermarks in the sand: Impossibility of strong watermarking for language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Jie Zhang, Debeshee Das, Gautam Kamath, and Florian Tramèr. Position: Membership inference attacks cannot prove that a model was trained on your data. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2025a. doi: 10.1109/SaTML64287.2025.00025.

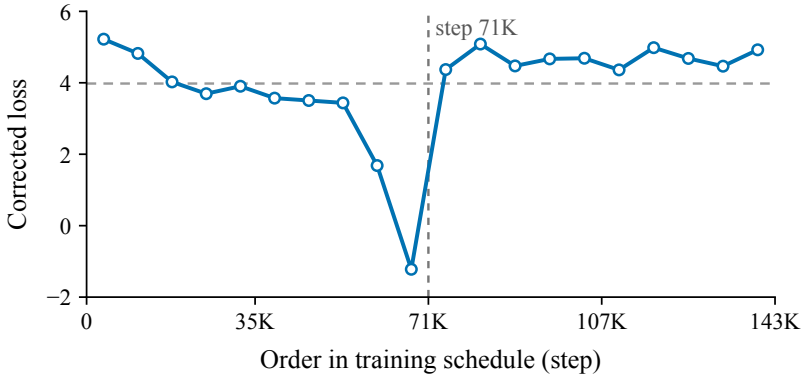
Jie Zhang, Dongrui Liu, Chen Qian, Linfeng Zhang, Yong Liu, Yu Qiao, and Jing Shao. REEF: Representation encoding fingerprints for large language models. In *International Conference on Learning Representations*, 2025b.

Rui Zhang, Jian Liu, Yuan Ding, Zhibo Wang, Qingbiao Wu, and Kui Ren. "Adversarial Examples" for Proof-of-Learning. In *2022 IEEE Symposium on Security and Privacy*, 2022. doi: 10.1109/SP46214.2022.9833596.

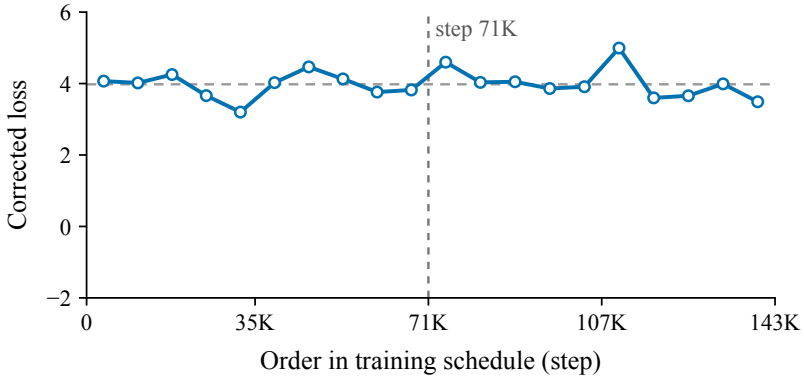
Sally Zhu, Ahmed M Ahmed, Rohith Kuditipudi, and Percy Liang. Independence tests for language models. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

A More on Palimpsestic Memorization

For Pythia, *Pythia* and *Pythia-deduped* refer to variants trained on the standard and deduplicated versions of the Pile, respectively. We use the latter only as a fixed loss reference. Figure 4 separates two effects visible at the step-71K checkpoint. Sequences encountered before the checkpoint have lower mean corrected loss than those not yet encountered (3.29 versus 4.67 nats per sequence), showing memorization of the used data. Within the used portion, sequences encountered later have lower mean corrected loss than earlier ones (2.20 versus 4.36 nats per sequence), showing that the model also retains information about training order. Randomly reordering the same corrected losses removes this order-aligned pattern. The random-order panel illustrates the comparison; statistical inference uses the registered permutation test.



(a) Pythia training order.



(b) Random order.

Figure 4: **Memorization and training-order dependence at the Pythia-6.9B step-71K checkpoint.** We evaluate 10,000 sequence occurrences sampled across the 143K-step training schedule. Corrected loss is the target model's sequence loss minus that of a fixed final Pythia-6.9B-deduped reference model; lower values indicate better prediction by the target relative to the reference. Points show means in 20 intervals, with ten equal-width intervals on each side of step 71K, under the original training order (top) and a random ordering of the same losses (bottom). The vertical dashed line marks the checkpoint, and the horizontal dashed line marks the mean over all sampled occurrences.

A.1 ADAPTIVE RETRAINING ADVERSARY

We consider an adaptive signed-gradient remover. At each step, it selects an example from its attack set, possibly one selected before, then chooses a signed step size based on the current model and previous updates.

Algorithm 1: Adaptive signed-gradient retraining adversary

Given. Fix initial parameters $\theta_0 \in \mathbb{R}^m$, an attack set $\mathcal{C}$ of candidate examples, a number of steps K, a loss function ℓ, and an aggregate coefficient cap $\bar{\eta}$.
For $j = 0, \ldots, K-1$**:**

1. The adversary adaptively selects a candidate $x_{i_j} \in \mathcal{C}$, possibly one selected before. The choice may depend on the current model θ_j and all previous updates.
2. The adversary chooses a signed step size $\eta_j \in \mathbb{R}$.
3. Apply the signed-gradient update

$$\theta_{j+1} = \theta_j - \eta_j \nabla_\theta \ell(\theta_j; x_{i_j}).$$

Positive η_j corresponds to learning and negative η_j to unlearning.

Output. The final parameters θ_K, subject to $|c_i| \leq \bar{\eta}$ for every $x_i \in \mathcal{C}$, where $c_i := \sum_{j:i_j=i} \eta_j$ is the aggregate signed coefficient of example x_i. The same example may be selected multiple times. The parameters are produced by the adaptively selected examples and signed step sizes.

Difference from the frozen-gradient adversary. The adaptive retraining adversary above evaluates each selected example's gradient at the current model θ_j. The frozen-gradient adversary retains the same adaptive choices of x_{i_j} and η_j, but evaluates every gradient at the initial model θ_0.

Algorithm 2: Frozen-gradient adaptive signed-gradient adversary

Given. Fix initial parameters $\theta_0 \in \mathbb{R}^m$, an attack set $\mathcal{C}$ of candidate examples, a number of steps K, a loss function ℓ, and an aggregate coefficient cap $\bar{\eta}$.
For $j = 0, \ldots, K-1$**:**

1. The adversary adaptively selects a candidate $x_{i_j} \in \mathcal{C}$, possibly one selected before. The choice may depend on the current model θ_j and all previous updates.
2. The adversary chooses a signed step size $\eta_j \in \mathbb{R}$.
3. Apply the signed frozen-gradient update

$$\theta_{j+1} = \theta_j - \eta_j \nabla_\theta \ell(\theta_0; x_{i_j}).$$

Positive η_j corresponds to learning and negative η_j to unlearning.

Output. The final parameters θ_K, subject to $|c_i| \leq \bar{\eta}$ for every $x_i \in \mathcal{C}$, where $c_i := \sum_{j:i_j=i} \eta_j$ is the aggregate signed coefficient of example x_i. The same example may be selected multiple times. The parameters are produced by the adaptively selected examples and signed step sizes using gradients evaluated at θ_0.

Additional evidence for the frozen-gradient approximation. The neural tangent kernel changes most rapidly early in training and then stabilizes (Fort et al., 2020). Recent work also finds that linearization can explain large-language-model fine-tuning (Afzal et al., 2026), and analyzes low-rank fine-tuning in the NTK regime (Jang et al., 2024). Related linearized and tangent-space methods have also been effective for fine-tuning or editing pretrained vision models (Achille et al., 2020; Ortiz-Jimenez et al., 2023). These results provide additional motivation for the frozen-gradient approximation; they do not establish its accuracy uniformly over every model or update region.

A.2 SUPPLEMENTARY EXPERIMENT RESULTS

Figure 6 examines how attack-set size affects the measured utility cost of reaching a non-significant audit result while holding the audit panel at 10K examples. For all three Pythia models, the relative WikiText perplexity increase is lower with 5K attack examples than with 500. The additional change

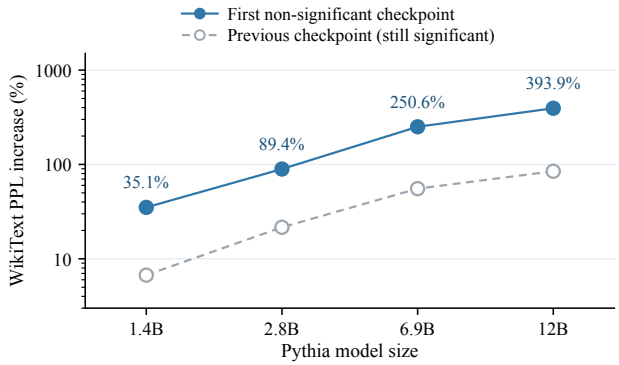


Figure 5: WikiText perplexity increase at signal-removal checkpoints across Pythia model sizes. Filled markers show the first tested checkpoint where the ownership test no longer detects the training-order signal; open markers show the immediately preceding checkpoint where the signal is still detected. The vertical axis shows the percentage increase in WikiText perplexity relative to each unmodified model, on a logarithmic scale. For each model, the attack set and audit panel contain the same 10,000 sampled sequences. Lines connect the tested model sizes.

from 5K to 10K is small at the measured replay checkpoints, suggesting diminishing gains over the attack-set sizes tested.

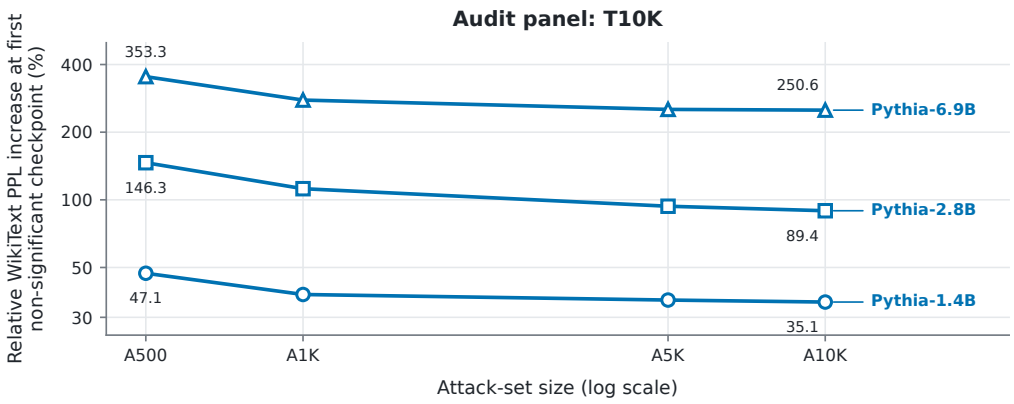


Figure 6: **Attack-set size and measured removal cost under a 10K audit panel.** For each Pythia model and attack-set size, a point reports the relative WikiText perplexity increase, $100(\mathrm{PPL}_{\mathrm{attacked}}/\mathrm{PPL}_{\mathrm{baseline}} - 1)$, at the first measured replay checkpoint satisfying $p_{\mathrm{perm}} > 0.01$. Baseline WikiText perplexities are 29.61, 25.97, and 23.69 for Pythia-1.4B, Pythia-2.8B, and Pythia-6.9B, respectively. Lines connect measured settings only.

Figure 7 examines downstream accuracy at the removal endpoints for Pythia-1.4B, 2.8B, and 6.9B. Accuracy decreases in all 12 model–benchmark comparisons. The largest declines are 19.4 percentage points on ARC-Easy and 17.7 points on HellaSwag for Pythia-6.9B, while WinoGrande declines by 1.6–2.9 points across the three models. Benchmark accuracy is measured after selecting each replay checkpoint using the audit result; it does not determine the selected checkpoint.

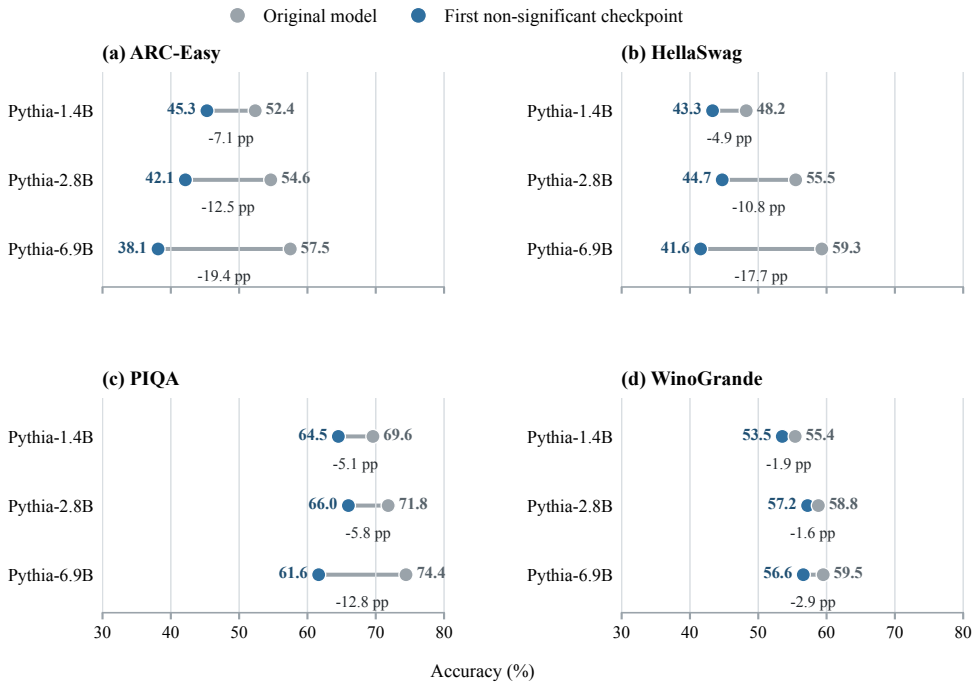


Figure 7: **Downstream accuracy at the first measured non-significant replay checkpoint.** For each Pythia model, the attack set is the entire audit panel of 10,000 sampled sequence occurrences. Gray points show the original model; blue points show its first measured replay checkpoint satisfying $p_{\text{perm}} > 0.01$. Point labels give zero-shot accuracy in percent, and annotations give the change in percentage points. ARC-Easy, HellaSwag, and PIQA use normalized accuracy; WinoGrande uses accuracy.

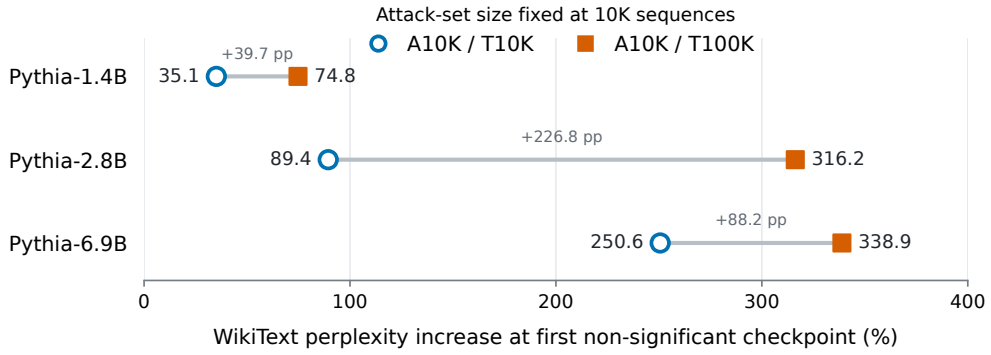


Figure 8: **Within-model comparison of removal cost under independently sampled A10K/T10K and A10K/T100K configurations.** Each row compares the two independently sampled audit-panel configurations for one Pythia model while holding the attack-set size fixed at 10K examples. Points report the WikiText perplexity increase at the first measured replay checkpoint satisfying $p_{\text{perm}} > 0.01$; gray segments connect the two configurations, and annotations report their difference. Reaching non-significance is associated with a larger perplexity increase under T100K for all three models.

B CONTESTABLE OWNERSHIP PROOFS

B.1 HOW TO BUILD CONTESTABLE OWNERSHIP PROOFS GENERICALLY

CRYPTOGRAPHIC PRIMITIVES

We formalize the public append-only ledger of Section 4 as an authenticated append-only bulletin board. We use this primitive to construct a generic contestable ownership proof from any of the three testing paradigms introduced in Section 2: secret-key fingerprints, training-data testing, and post-hoc relatedness testing. We use the bulletin board model (see, e.g. Choudhuri et al. (2017) and Shi (2026)). In each of the following subsections, we then formalize one of these three notions as an interface, and provide a generic construction of contestable ownership proof from them. This section is of both theoretical and practical interest, and provide pointers for future constructions.

We focus on a decentralized realization of the bulletin board, because it offers an interface a centralized board does not: a public beacon that releases a value no party could have predicted or biased in advance. Our construction uses this beacon to fix the order of training data only after an owner's commitment is already final, which is what stops a dishonest owner from choosing a favorable order after seeing the data. A centralized board, such as a certificate-transparency log, gives append-only, publicly checkable publication but no such beacon. A decentralized board realized from a blockchain, such as Ethereum under proof of stake, gives both.

Definition 1 (Authenticated Append-only Bulletin board)**.** An append-only bulletin board is a tuple of probabilistic polynomial-time algorithms $(\mathsf{BB.Setup}, \mathsf{BB.Publish}, \mathsf{BB.Read}, \mathsf{BB.Beacon})$. It is public: any party may call the algorithms with the same inputs and receive the same outputs. We follow the bulletin-board model of (Choudhuri et al., 2017; Shi, 2026).

- $\mathsf{BB.Setup}(1^\lambda)$ outputs public parameters and gives each registered user a public identifier id (its authentication public key) and a private authentication key $\mathsf{sk_{id}}$.
- $\mathsf{BB.Publish}(\mathsf{sk_{id}}, m)$ authenticates the publisher and appends (id, m) to the board. It returns the record's index i. We identify record indices with beacon rounds: record i is finalized before $\mathsf{BB.Beacon}(i+1)$ is revealed.
- $\mathsf{BB.Read}(i)$ returns the identity and message (id, m) stored at index i, or $\perp$ if no such record exists.
- $\mathsf{BB.Beacon}(\tau)$ takes a round τ and, once round τ has been reached, returns a public random string.

The bulletin board satisfies the following properties.

1. **Unforgeability.** Once $\mathsf{BB.Publish}(\mathsf{sk_{id}}, m)$ returns i, the record (id, m) at index i is fixed. For every probabilistic polynomial-time adversary with oracle access to $\mathsf{BB.Publish}$ and $\mathsf{BB.Read}$, the probability that a later call to $\mathsf{BB.Read}(i)$ returns (id, m') for any $m' \neq m$ is at most $\mathrm{negl}(\lambda)$.
2. **Unpredictability.** Fix a round τ and let $\mathsf{view}_{\tau-1}$ be everything any party could have published or read up to round $\tau - 1$. For every probabilistic polynomial-time adversary Adv and uniform r over the beacon's output space,

   $$|\Pr[\mathsf{Adv}(\mathsf{view}_{\tau-1}, \mathsf{BB.Beacon}(\tau)) = 1] - \Pr[\mathsf{Adv}(\mathsf{view}_{\tau-1}, r) = 1]| \leq \mathrm{negl}(\lambda).$$

   In particular, no adversary can predict $\mathsf{BB.Beacon}(\tau)$, or bias it toward a chosen value, using information available before round τ.
3. **Authentication.** For every probabilistic polynomial-time adversary, the probability of causing $\mathsf{BB.Read}(i)$ to return (id, m) when m was not submitted using the authentication key associated with id is at most $\mathrm{negl}(\lambda)$.

For an execution of the bulletin board, a party's total publication cost is the sum of $\mathsf{BB.PublishCost}(i)$ over the indices assigned to its successful calls to $\mathsf{BB.Publish}$.

Definition 2 (Vector commitment)**.** A vector commitment scheme consists of probabilistic polynomial-time algorithms

$$(\mathsf{VC.Gen}, \mathsf{VC.Digest}, \mathsf{VC.Open}, \mathsf{VC.Vf}).$$

The setup algorithm $\mathsf{VC.Gen}(1^\lambda)$ initializes public parameters available to all parties. We leave these parameters implicit in the remaining algorithms.

- $\mathsf{VC.Digest}(X)$ takes a data corpus in fixed canonical order, $X = (x_1, \ldots, x_N)$ and outputs a commitment com and auxiliary state st. The commitment binds both the entries and the length of X.
- $\mathsf{VC.Open}(\mathsf{st}, Q)$ takes a set of positions $Q \subseteq [N]$ and outputs an opening op for the values $X[Q]$.
- $\mathsf{VC.Vf}(com, Q, Y, \mathsf{op}) \in \{0, 1\}$ checks whether op opens the positions Q in the committed vector to the values Y.

The scheme satisfies the following properties.

Correctness. For every finite vector X, let

$$(com, \mathsf{st}) \leftarrow \mathsf{VC.Digest}(X).$$

For every $Q \subseteq [|X|]$, if

$$\mathsf{op} \leftarrow \mathsf{VC.Open}(\mathsf{st}, Q),$$

then

$$\Pr[\mathsf{VC.Vf}(com, Q, X[Q], \mathsf{op}) = 1] = 1.$$

Binding. For every probabilistic polynomial-time adversary Adv run after $\mathsf{VC.Gen}$, the probability of producing two accepted openings under the same commitment that disagree at a shared position is negligible. Write $Y[i]$ for the value associated with queried position i in an opening. Formally,

$$\Pr\left[\begin{array}{c}(com, Q, Q', Y, Y', \mathsf{op}, \mathsf{op}') \leftarrow \mathsf{Adv}: \\ \mathsf{VC.Vf}(com, Q, Y, \mathsf{op}) = 1 \\ \wedge\ \mathsf{VC.Vf}(com, Q', Y', \mathsf{op}') = 1 \\ \wedge\ \exists i \in Q \cap Q' :\ Y[i] \neq Y'[i]\end{array}\right] \leq \mathrm{negl}(\lambda).$$

The probability is over the randomness of $\mathsf{VC.Gen}$ and Adv. The query sets may differ: binding requires their accepted answers to agree at every shared position. This property does not by itself provide an algorithm that extracts the committed vector from a malicious commitment.

Hiding. Consider two equal-length vectors X_0 and X_1. A challenger samples $b \leftarrow \{0, 1\}$, computes

$$(com, \mathsf{st}) \leftarrow \mathsf{VC.Digest}(X_b),$$

and gives com to the adversary. The adversary may request an opening for any set Q such that $X_0[Q] = X_1[Q]$, and receives

$$\mathsf{VC.Open}(\mathsf{st}, Q).$$

For every probabilistic polynomial-time adversary that outputs a guess b',

$$\left|\Pr[b' = b] - \frac{1}{2}\right| \leq \mathrm{negl}(\lambda).$$

The adversary therefore cannot determine which equal-length vector was committed with more than negligible advantage over random guessing. The condition $X_0[Q] = X_1[Q]$ ensures that requested openings do not reveal b directly; subject to those openings, the commitment hides the unopened entries.

B.1.1 CONTESTABLE OWNERSHIP PROOF FROM A FINGERPRINT TEST

Definition 3 (Fingerprint test)**.** Fix an admissible corpus family $\mathcal{X}$, a model space Θ, a key space $\mathcal{K}$, and a constant $\delta \in (0, 1/2)$. A fingerprint test consists of three algorithms $(\mathsf{Setup}_{\mathrm{FT}}, \mathsf{Train}_{\mathrm{FP}}, \mathsf{Detect})$:

- **Setup.** $\mathsf{Setup}_{\mathrm{FT}}(1^\lambda)$ takes the security parameter and outputs a key $K \in \mathcal{K}$.
- **Training.** $\mathsf{Train}_{\mathrm{FP}}(\theta_0, X, K)$ takes an initial model θ_0, a data corpus X, and a key K. It outputs a trained model θ.
- **Detection.** $\mathsf{Detect}^X(\theta, K; r) \in \{0, 1\}$ takes a model θ, a key K, and randomness r, with read-only oracle access to X. It returns one if it detects the fingerprint associated with K in θ, and zero otherwise.

For a corpus $X = (x_1, \ldots, x_N)$, oracle access to X allows a query i to return the occurrence x_i. The algorithms have the following requirements:

1. **Bounded false negatives.** For every corpus $X \in \mathcal{X}$, initial model $\theta_0 \in \Theta$, and probabilistic polynomial-time adversary, let $K \leftarrow \mathsf{Setup}_{\mathrm{FT}}(1^\lambda)$ and $\theta \leftarrow \mathsf{Train}_{\mathrm{FP}}(\theta_0, X, K)$. The adversary chooses an admissible strategy γ and produces $\theta^\star \leftarrow \mathcal{A}_\gamma(\theta)$. With probability at least $1-\varepsilon$ over setup, training, and the adversary's randomness, including the modification's randomness, $\Pr_r[\mathsf{Detect}^X(\theta^\star, K; r) = 0] \le \delta$.
2. **Bounded false positives under further training.** Fix an initial model $\theta_0 \in \Theta$ and a corpus $X \in \mathcal{X}$. Sample $K' \leftarrow \mathsf{Setup}_{\mathrm{FT}}(1^\lambda)$ and compute $\theta \leftarrow \mathsf{Train}_{\mathrm{FP}}(\theta_0, X, K')$. Independently sample the old key $K \leftarrow \mathsf{Setup}_{\mathrm{FT}}(1^\lambda)$; it is not supplied to the training process. With probability at least $1 - \varepsilon$ over both key generations and training, $\Pr_r[\mathsf{Detect}^X(\theta, K; r) = 1] \le \delta$.
3. **Bounded false positives at initialization.** For every corpus $X \in \mathcal{X}$, independently sample θ_0 from the initialization distribution and $K \leftarrow \mathsf{Setup}_{\mathrm{FT}}(1^\lambda)$. With probability at least $1 - \varepsilon$ over initialization and key generation, $\Pr_r[\mathsf{Detect}^X(\theta_0, K; r) = 1] \le \delta$.

For fixed X, θ, and K, the inner probability is over the detector randomness r; repeated executions use independent randomness.

Throughout this construction, $X = (x_1, \ldots, x_N)$ denotes the full data corpus in fixed canonical order, and com denotes its cryptographic commitment, as in Section 4. The ordered examples used for training are called the training data.

A predecessor is an earlier released model used to obtain θ_0, directly or through further training. The declaration $\mathcal{P}$ lists the claims of acknowledged predecessors.

Records have fixed board indices; hash inputs are unambiguously encoded. The beacon output r supplies randomness for a single detector execution. For the loss–order instantiation, Train and $\mathsf{VerifyProof}$ also require $2 \le \lfloor \alpha N \rfloor < N$ and $2 \le n \le N$.

Protocol 1: Contestable ownership from a fingerprint test

Public ingredients and setup: Use the bulletin board, vector commitment, and fingerprint test of Definitions 1–3, and a random-oracle hash H. $\mathsf{Setup}(1^\lambda)$ runs $\mathsf{BB.Setup}(1^\lambda)$ and $\mathsf{VC.Gen}(1^\lambda)$, fixes a positive integer n for the audit-panel size, and returns public parameters pp. For each training registration, the generic fingerprint key K is generated by Train after that registration is finalized. Each registered user has a unique public identifier id under which the bulletin board authenticates its records.

$\mathsf{Train}(\mathsf{id}, X, \alpha, \theta_0, \mathcal{P})$

Here $\alpha \in (0, 1]$ is the training fraction.

1. **Register the training information.** Set $N = |X|$ and reject unless $0 < \alpha \leq 1$ and $1 \leq \lfloor \alpha N \rfloor \leq N$. If $\mathcal{P} = \varnothing$, sample θ_0 from the initialization distribution before obtaining the training beacon randomness. Compute $(com, \mathsf{st}_X) \leftarrow \mathsf{VC.Digest}(X)$, publish $\mathsf{Reg} = (\mathsf{id}, com, N, \alpha, H(\theta_0), \mathcal{P})$, and let j_{reg} be its board index.
2. **Generate the fingerprint key and training data order.** After the registration is finalized, obtain $r_{\mathrm{train}} \leftarrow \mathsf{BB.Beacon}(j_{\mathrm{reg}} + 1)$. Run $\mathsf{Setup}_{\mathrm{FT}}(1^\lambda)$ with a public random tape obtained from counter-tagged calls to H on $(\mathtt{key}, \mathsf{id}, com, r_{\mathrm{train}})$, and denote its output by K. The counter supplies fresh bits as needed, so any party can reproduce K. For the loss–order test, use K to select and order $\lfloor \alpha N \rfloor$ occurrences from X by the public rule in Section B.1.2.
3. **Train and release.** Compute $\theta \leftarrow \mathsf{Train}_{\mathrm{FP}}(\theta_0, X, K)$ and publish $\mathsf{Rel} = (j_{\mathrm{reg}}, H(\theta))$. Let j_{rel} be its board index, set $c = (j_{\mathrm{reg}}, j_{\mathrm{rel}})$ and $\mathsf{st} = (X, \mathsf{st}_X)$, and return (c, θ, st).

$\mathsf{Prove}(c, c_\theta, \theta, \mathsf{st})$

1. **Obtain audit randomness.** Let j_{rel} be the target model's release index in c_θ. Obtain $r \leftarrow \mathsf{BB.Beacon}(j_{\mathrm{rel}} + 1)$.
2. **Collect the detector queries.** Read (X, st_X) from st and the source registration from c; let j_{reg} be its index. Recompute K from that registration and $\mathsf{BB.Beacon}(j_{\mathrm{reg}} + 1)$ using the key-generation rule in Train. Simulate $\mathsf{Detect}^X(\theta, K; r)$ and let Q be the set of corpus indices it queries.
3. **Open and return.** Compute $\mathsf{op}_Q \leftarrow \mathsf{VC.Open}(\mathsf{st}_X, Q)$ and return $\Pi = (Q, X[Q], \mathsf{op}_Q)$.

$\mathsf{VerifyProof}(c, c_\theta, \theta, \Pi)$

1. **Validate the commitment.** Parse $\Pi = (Q, X[Q], \mathsf{op}_Q)$; reject malformed inputs. Read the authenticated registration $(\mathsf{id}, com, N, \alpha, H(\theta_0), \mathcal{P})$ on the bulletin board identified by c, and let j_{reg} be its index. Recompute K from that registration and $\mathsf{BB.Beacon}(j_{\mathrm{reg}} + 1)$ using the key-generation rule in Train. Check $0 < \alpha \leq 1$ and $1 \leq \lfloor \alpha N \rfloor \leq N$, $K \in \mathcal{K}$, and that the release in c is authentic, was published under id after the training beacon was revealed, and references the registration index. Reject unless c_θ identifies an authenticated release with model digest $H(\theta)$; let j_{rel} be its index. Check that the source release precedes the revelation of $\mathsf{BB.Beacon}(j_{\mathrm{rel}} + 1)$.
2. **Reproduce the queries.** Obtain $r \leftarrow \mathsf{BB.Beacon}(j_{\mathrm{rel}} + 1)$. Simulate $\mathsf{Detect}^X(\theta, K; r)$ using the supplied $X[Q]$; reject any query outside Q. Record the detector output as A and check that its queried indices are exactly Q.
3. **Verify the commitment proof.** Check $Q \subseteq [N]$ and $\mathsf{VC.Vf}(com, Q, X[Q], \mathsf{op}_Q) = 1$.
4. **Accept or reject.** Return 1 if all checks pass and $A = 1$; otherwise return 0.

$\mathsf{Challenge}(c, c_\theta, \theta, \mathsf{st})$

Generate the challenge. Return $\Pi_{\mathrm{ch}} \leftarrow \mathsf{Prove}(c, c_\theta, \theta, \mathsf{st})$.

$\mathsf{Judge}(c, c_\theta, \theta, \mathcal{P}_{c_\theta}, \Pi_{\mathrm{ch}})$

1. **Validate the release order.** Reject if c_θ does not identify an authenticated release with model digest $H(\theta)$, or if that release does not follow the release in c.
2. **Check the omission.** Reject if c occurs in $\mathcal{P}_{c_\theta}$.

3. **Verify the ownership proofs and challenge.** Obtain the source model θ_c and the published self-proofs Π_c and Π_{c_θ}; reject missing or malformed submissions. Reject unless $\mathsf{VerifyProof}(c, c, \theta_c, \Pi_c) = 1$ and $\mathsf{VerifyProof}(c_\theta, c_\theta, \theta, \Pi_{c_\theta}) = 1$. Return $\mathsf{VerifyProof}(c, c_\theta, \theta, \Pi_{\mathrm{ch}})$.

B.1.2 Instantiating the fingerprint test with a model-memorization fingerprint

Protocol 2: Model-memorization fingerprint with Pearson detection

Public inputs and parameters: The algorithms use a training fraction $\alpha \in (0, 1)$ with $2 \le \lfloor \alpha N \rfloor < N$, query-set size $2 \le n \le N$, reference model $\theta_{\mathrm{ref}} \in \Theta$, significance level $a \in (0, 1/2)$, and number R of random permutations. We use the two-sided Pearson permutation test below throughout this paper, with $a = 0.01$ in the experiments.

$\mathsf{Setup}_{\mathrm{FT}}(1^\lambda)$

Sample a uniform seed $K \in \{0,1\}^\lambda$ and return K. When used in Protocol 1, its λ random bits are supplied by the public key-generation rule in Train after registration finalization.

$\mathsf{Train}_{\mathrm{FP}}(\theta_0, X, K)$

Use K and training-domain calls to the public random oracle H to generate a random permutation of the $N = |X|$ corpus indices.[a] Let D be the ordered sequence of occurrences at its first $\lfloor \alpha N \rfloor$ indices, train from θ_0 on D in that order, and return the resulting model θ. If occurrence i is the jth element of D, set $o_i = j$; otherwise set $o_i = 0$.

$\mathsf{Detect}^X(\theta, K; r)$

1. Reconstruct the positions o_i from K, N, α. Using fresh bits of r, sample a uniform n-element query set $Q \subseteq [N]$ and query $X(i)$ for every $i \in Q$.
2. For every $i \in Q$, compute the corrected loss $y_i = L(\theta; X(i)) - L(\theta_{\mathrm{ref}}; X(i))$. Return zero if a loss is nonfinite or if either sampled vector has zero variance.
3. Let o_Q and y_Q denote the training-coordinate and corrected-loss vectors on Q. The detector computes their two-sided Pearson permutation p-value. Using fresh bits of r, it approximates the permutation null distribution with R independent uniform permutations π_b of the entries of o_Q, including the zeros, while keeping y_Q fixed:
$$p_{\mathrm{perm}} = \frac{1 + \sum_{b=1}^{R} \mathbf{1}\{|\operatorname{Corr}(\pi_b(o_Q), y_Q)| \ge |\operatorname{Corr}(o_Q, y_Q)|\}}{R+1}.$$
4. Return one if $p_{\mathrm{perm}} \le a$, and zero otherwise.

[a] One implementation expands oracle inputs beginning with $(\mathtt{train}, K, N, \lfloor \alpha N \rfloor)$ into N independent labels of $\lambda + 2\lceil \log_2 N \rceil$ bits, and sorts the indices by their labels, breaking ties by index. Label collisions have probability at most $2^{-\lambda}$; conditional on no collision, the permutation is uniform.

AI Use Statement All research concepts, core protocol logic, and experimental designs were developed by the authors. LLMs were used to assist with the development and debugging of scripts for our experimental framework, drafting and polishing parts of the manuscript, and identifying relevant literature.