electronics

Article

AuditableLLM: A Hash-Chain-Backed, Compliance-Aware Auditable Framework for Large Language Models

Delong Li [1], Guangsheng Yu [1,*], Xu Wang [1] and Bin Liang [2]

[1] School of Electrical & Data Engineering, University of Technology Sydney, Ultimo, NSW 2007, Australia
[2] Data Science Institute, University of Technology Sydney, Ultimo, NSW 2007, Australia
* Correspondence: guangsheng.yu@uts.edu.au

Abstract

Auditability and regulatory compliance are increasingly required for deploying large language models (LLMs). Prior work typically targets isolated stages such as training or unlearning and lacks a unified mechanism for verifiable accountability across model updates. This paper presents AuditableLLM, a lightweight framework that decouples update execution from an audit-and-verification layer and records each update as a hash-chain-backed, tamper-evident audit trail. The framework supports parameter-efficient fine-tuning such as Low-Rank Adaptation (LoRA) and Quantized LoRA (QLoRA), full-parameter optimization, continual learning, and data unlearning, enabling third-party verification without access to model internals or raw logs. Experiments on LLaMA-family models with LoRA adapters and the MovieLens dataset show negligible utility degradation (below 0.2% in accuracy and macro-F1) with modest overhead (3.4 ms/step; 5.7% slowdown) and sub-second audit validation in the evaluated setting. Under a simple loss-based membership inference attack on the forget set, the audit layer does not increase membership leakage relative to the underlying unlearning algorithm. Overall, the results indicate that hash-chain-backed audit logging can be integrated into practical LLM adaptation, update, and unlearning workflows with low overhead and verifiable integrity.

Keywords: auditable learning framework; large language models; hash-chain; compliance-aware AI; LoRA; unlearning; verification

1. Introduction

Large language models (LLMs) have rapidly evolved from experimental research artifacts into foundational infrastructure across safety-critical domains. As their capabilities expand, so too do expectations for transparency, accountability, and regulatory compliance in how such models are trained, adapted, and maintained. Frameworks such as the EU's General Data Protection Regulation (GDPR) and emerging AI governance acts require developers to demonstrate data provenance, model traceability, and responsible update management throughout the model lifecycle [1–6]. Meeting these obligations demands not only privacy-preserving algorithms but also verifiable audit mechanisms that can attest to how model parameters evolve over time.

Recent advances in large-scale adaptation have led to parameter-efficient fine-tuning (PEFT) methods such as adapters, Low-Rank Adaptation (LoRA), and QLoRA, enabling efficient specialization of LLMs without full retraining [7–9]. At the same time, emerging research in continual learning and machine unlearning seeks to support principled knowledge addition and removal [10–14]. Despite addressing different objectives, these research

Check for updates

Academic Editor: Dimitris Apostolou

Received: 22 November 2025
Revised: 17 December 2025
Accepted: 19 December 2025
Published: 23 December 2025

directions share a critical gap: the lack of a unified and verifiable auditability mechanism for tracking and attesting model update operations. Existing approaches often rely on empirical observations or heavyweight cryptographic verification, including proof-of-learning (PoL) techniques that attempt to attest legitimate training but face scalability and robustness limitations in practice [15,16], making them difficult to integrate into real-world LLM training pipelines or Machine-Learning-as-a-Service (MLaaS) deployments. In parallel, provenance-aware logging and model governance frameworks typically focus on high-level documentation and data lineage rather than fine-grained, tamper-evident tracking of concrete update operations. At the same time, unlearning methods are usually designed around specific algorithms or threat models and rarely expose a reusable audit interface across adaptation regimes. Taken together, these limitations leave practitioners without a lightweight, algorithm-agnostic audit layer for parameter-efficient LLM adaptation that spans fine-tuning, continual updates, and compliance-driven deletions.

In response to this gap, AuditableLLM offers a lightweight audit framework for LLMs that sits at the intersection of verifiable training and proofs-of-learning, provenance-aware logging and model governance, and machine unlearning. The framework decouples model update operations from the audit and verification layer and is designed to augment a broad class of learning and unlearning processes with a hash-chain-backed, tamper-evident trail. This design provides transparency into how model parameters evolve without exposing raw data or proprietary internals, thereby supporting compliance-aware accountability under the stated threat model. The empirical study serves as a proof-of-concept validation of audit-layer properties (tamper-evidence, traceability, and overhead) under a controlled adaptation workflow. Under the stated threat model and logging protocol, the audit and verification mechanism remains model- and task-agnostic: its audit and verification complexity is governed primarily by the number of logged update events (i.e., the update-chain length T), which supports scalability to larger backbones and longer update pipelines.

The main contributions are as follows:

- This work designs a unified audit pipeline that supports diverse model update mechanisms, including full-parameter fine-tuning, parameter-efficient fine-tuning (PEFT), and unlearning, while preserving task utility and model stability in the LoRA-based experimental setting.
- This work develops an audit layer that records verifiable, hash-chained logs of update events and metadata, enabling third-party verification without costly cryptographic primitives.
- This work introduces a multi-level verification suite that combines behavioral, parametric, and risk-oriented diagnostics to assess consistency and compliance properties of the audited adaptation process.
- This work demonstrates the practicality of the framework through experiments on LLM adaptation and data deletion tasks using the LLaMA model family with LoRA adapters and the MovieLens dataset.

AuditableLLM is positioned within the broader research landscape of verifiable and accountable AI [4]. By bridging the gap between efficient model adaptation and formal auditability in this honest-but-curious, small-to-mid-scale setting, the proposed framework illustrates how a unified foundation for transparent, compliance-aware, and auditable LLM adaptation can be constructed and evaluated. The remainder of this paper is organized as follows: Section 2 reviews background and related work; Section 3 presents the system model and design objectives; Section 4 details the proposed methodology and system architecture; Section 5 reports experimental results and analyses; and Section 6 concludes the paper with future directions and policy implications.

2. Background and Related Work

This section is divided into Background and Related Work. The Background introduces core concepts in LLM adaptation, continual learning, unlearning, and hash-chain auditing, while the Related Work reviews existing approaches in verifiable training and audit mechanisms, motivating the need for a unified auditable adaptation framework.

2.1. Background

2.1.1. LLM Adaptation and PEFT

LLMs are typically adapted to downstream tasks through update procedures that modify model parameters based on new data or objectives. Conventional full-parameter fine-tuning enables high task performance but incurs substantial computational and storage costs, which limits its practicality in iterative or resource-constrained settings. To address these challenges, PEFT has emerged as a dominant paradigm for LLM adaptation. PEFT methods update only a small subset of parameters while reusing the frozen pretrained backbone, achieving task specialization with reduced computational overhead.

Representative PEFT approaches include adapter modules [7,17], LoRA [8], and its quantization-optimized extension QLoRA [9]. These techniques introduce compact trainable components in place of full model updates, which leads to faster training, lower memory consumption, and improved modularity. Since parameter changes are confined to well-structured and low-dimensional subspaces, each adaptation becomes easier to track, store, and analyze throughout the model lifecycle. This modular structure naturally supports principled logging and traceability, providing a technical foundation for auditable model adaptation.

This work adopts LoRA-based fine-tuning as a representative adaptation mechanism when instantiating the proposed AuditableLLM framework. Although the experiments focus on PEFT for efficiency and clarity, the auditing principles apply broadly to any LLM update regime, including full fine-tuning, continual learning, or selective data removal. In essence, the audit layer operates independently of the training algorithm, treating every model update as a recordable and verifiable event.

2.1.2. Continual Learning and Machine Unlearning

As LLMs interact with evolving data sources and deployment environments, their parameters must be updated over time to incorporate new knowledge, adapt to shifting distributions, or refine model behavior. Continual learning provides a paradigm for achieving such incremental updates by allowing models to learn new tasks or data streams without retraining from scratch [18–20]. A central objective in continual learning is to avoid catastrophic forgetting, where learning new information causes previously acquired knowledge to degrade. Techniques in this area often introduce memory-aware optimization strategies, rehearsal modules, or regularization mechanisms to preserve model stability across updates.

Complementing continual adaptation, machine unlearning aims to remove the influence of specific data points or data subsets from a trained model [10,11]. Machine unlearning has gained increasing importance due to privacy regulations and user rights such as the right to erasure under modern legal frameworks. Unlike continual learning, which focuses on integrating new knowledge, machine unlearning addresses the reverse challenge of retracting learned information while preserving the utility of the remaining knowledge. Different machine unlearning strategies employ model retraining, parameter surgery, or approximation-based removal, depending on the sensitivity and scope of the target data.

Both continual learning and machine unlearning require reliable mechanisms to track how model parameters evolve over time. Each update, whether additive or subtractive, produces changes that affect model behavior, compliance status, and downstream reliability. From an auditability perspective, these update operations must be recorded in a manner that enables later inspection and verification. This motivates the need for auditable adaptation frameworks in which all update events can be traced, attributed, and validated throughout the lifecycle of an LLM.

2.1.3. Verifiable Training and Hash-Chain Auditing

As model updates accumulate over time, ensuring the integrity and provenance of each modification becomes essential for trustworthy deployment. Verifiable training seeks to provide evidence that a claimed update was executed as intended, producing parameter changes that are consistent with an admissible learning rule. This perspective treats model evolution as a sequence of authenticated steps, where each update can be inspected and validated to support accountability, reproducibility, and compliance throughout the lifecycle of an LLM.

Hash-chained logging offers a practical foundation for constructing tamper-evident records of model updates. By computing cryptographic digests for each update and linking them through hash pointers, hash-chain mechanisms create immutable audit trails in which any alteration of historical records becomes immediately detectable. This approach enables update provenance to be verified without exposing internal model states or raw training data. In contrast to full cryptographic proofs, hash-chain auditing supports efficient and incremental verification that scales with the number of recorded updates.

When applied to LLM adaptation and maintenance workflows, hash-chain auditing provides the means to track both additive operations such as training and continual learning, and subtractive operations such as machine unlearning. By binding update events to verifiable records, this mechanism enables transparent oversight of model evolution and establishes the technical foundation for auditable adaptation in real-world systems.

2.2. Related Work

2.2.1. Auditable and Verifiable Training

Early efforts toward verifiable training aim to provide evidence that model updates follow legitimate learning trajectories. PoL schemes represent a central line of work in this direction. Jia et al. introduced formal definitions and mechanisms that enable a model to produce admissible proofs of its training history [15]. These methods seek to establish verifiable accountability by binding parameter updates to provable computation traces. However, subsequent studies revealed that many PoL variants are vulnerable to adaptive attacks and replay strategies, which raises concerns regarding their reliability in practical deployment scenarios [16]. Despite their theoretical rigor, PoL-based approaches often introduce considerable computational overhead and remain challenging to integrate into large-scale LLM training pipelines.

Complementary work explores cryptographic verification using succinct proofs, such as systems based on SNARKs and related primitives [21]. These approaches offer strong integrity guarantees but require heavy proof generation and verification costs, which limit their applicability in frequent-update settings such as continual learning or fine-grained adapter tuning. As a result, while existing methods demonstrate that verifiable training is attainable in principle, they are not yet well aligned with the efficiency and scalability requirements of modern LLM adaptation workflows.

These observations highlight a persistent gap: current verifiable training techniques either provide strong guarantees at prohibitive cost or offer partial assurances without end-to-end coverage of diverse update types. This motivates the need for alternative auditing mechanisms that balance verifiability with practicality, particularly for dynamic LLM update regimes that involve both learning and unlearning.

Beyond PoL-style schemes, recent work has also explored zero-knowledge proofs of training (zkPoT) and related protocols that provide cryptographic evidence of correct training without revealing the underlying data or model parameters. Garg et al. [21], for example, demonstrate zkPoT constructions for machine learning models within the SNARK framework. Such ZK- and SNARK-based methods offer stronger, publicly verifiable guarantees than simple hash-chained logs, but currently incur substantial proving costs and scale only to relatively restricted model sizes and objectives. In this sense, they are complementary to lightweight audit layers such as AuditableLLM, which aim to provide efficient, tamper-evident recordkeeping and can, in principle, be combined with heavier proof systems in a layered attestation stack.

2.2.2. Auditable Logging, Blockchain, and FL-Based Auditing

Another line of research focuses on auditable logging and distributed accountability in learning systems. Blockchain-based approaches have been proposed to record model updates or gradients on immutable ledgers, providing tamper-resistant provenance for collaborative or distributed training settings [22]. These systems offer transparent traceability but inherit the computational and communication overhead of blockchain consensus protocols, which poses challenges for fast or frequent model updates.

Similarly, federated learning (FL) audit mechanisms rely on distributed trust assumptions to ensure that client-side updates can be attested and tracked [2,23,24]. While such methods improve accountability in multi-party environments, they primarily address cross-device or cross-silo verification rather than fine-grained update auditing within a single model lifecycle. As a result, they provide limited support for centralized adaptation workflows where updates occur at high frequency or originate from a single authority.

Logging-based systems that capture events, metrics, or gradient histories offer complementary auditing capabilities, yet they often depend on external storage and ad hoc verification tools rather than providing intrinsic guarantees within the update process. These approaches facilitate retrospective analysis but do not ensure that recorded updates are complete, tamper-resistant, or verifiable at the algorithmic level. Recent work on efficient and certified recovery from poisoning attacks in federated learning [25] further illustrates how structured logs of client updates can support post-hoc forensic analysis and rollback of malicious contributions. In a similar spirit, the hash-chain-backed audit layer in AuditableLLM provides a structured, tamper-evident log of adaptation steps that can be leveraged to attribute abnormal or malicious behavior to specific fine-tuning steps or unlearning requests in a centralized LLM setting, and to support rollback to known-good checkpoints. A promising direction for future work is to systematically combine such logs with poisoning-robust and certified recovery procedures, using the audit metadata to drive automated detection and remediation of poisoned updates.

Overall, blockchain and FL-based auditing techniques enhance transparency in distributed training scenarios but do not serve as general solutions for auditable model adaptation. They lack mechanisms for unified verification across heterogeneous update types such as fine-tuning and unlearning, and are not designed to integrate directly into local LLM update pipelines with efficient per-step attestations.

2.2.3. Unlearning Verification

A complementary direction investigates the verification of machine unlearning, where the goal is to assess whether a model has successfully removed information associated with specific data. Existing approaches often rely on behavioral or statistical proxies that measure how the model responds to retained and forgotten samples [10,11]. These diagnostics can reveal residual influence or incomplete forgetting, but they provide indirect evidence and do not trace the underlying update process. As a result, they cannot guarantee that a claimed unlearning operation was executed in full or in accordance with a prescribed update rule.

Some recent methods further incorporate influence-based metrics or membership inference tools to quantify whether sensitive data remains encoded in model parameters [26,27]. Although these techniques enhance empirical assessment, they focus on outcomes rather than the correctness or integrity of the update procedure itself. They also depend on probabilistic judgments or adversarial testing, which can lead to ambiguous conclusions in deployment settings.

Overall, unlearning verification research has advanced techniques for evaluating the effects of data removal, but current methods lack a mechanism to authenticate the unlearning process or record it in a verifiable manner. These limitations reflect the absence of an integrated auditing substrate that can attest update provenance, enforce accountability, and support compliance guarantees across diverse modification types.

2.2.4. Membership Inference Attacks and Machine Unlearning

Membership inference attacks (MIAs) [28,29] exploit the fact that trained models often assign systematically lower loss or higher confidence to training points than to unseen examples. In white-box settings, per-example training loss and gradient information can serve as powerful membership signals, enabling an adversary to infer whether a particular data point was used during training or fine-tuning. Recent work has begun to combine MIAs with unlearning evaluations, using the attack success rate as an indicator of how much information about deleted data remains encoded in model parameters.

This work adopts a simple loss-based MIA in the spirit of prior attacks [28–30] to compare the membership leakage of fully trained, retrained, and unlearned models on the MovieLens-small task. The analysis focuses on whether integrating a hash-chain-backed audit layer changes the membership privacy profile of an existing parameter-efficient unlearning algorithm, rather than on designing new MIAs.

2.2.5. Comparison with the Proposed Framework

The three research directions reviewed above highlight complementary yet fragmented progress toward accountable model adaptation. Cryptographic verification and PoL methods offer strong guarantees but incur substantial computational cost, which limits their deployment in large-scale update pipelines. Blockchain and federated logging frameworks provide tamper-resistant provenance, yet they are designed for distributed trust rather than fine-grained auditing of local model updates. Unlearning verification techniques focus on assessing the outcomes of data removal, but they do not authenticate the update process itself or ensure the integrity of its execution. Because these assurance paradigms certify different properties under different threat assumptions and operational constraints, this work does not treat their costs as directly comparable under a single uniform metric. Instead, the comparison explicitly distinguishes (i) what is being certified (training correctness, unlearning correctness, or update-history integrity), (ii) the assumed adversary/trust model, and (iii) the deployment requirements. The complexity of AuditableLLM is analyzed separately in Section 4.4 to avoid conflating heterogeneous cost models across paradigms.

In contrast, this work introduces a unified audit framework that verifies training, adaptation, and unlearning through a single audit layer. Rather than depending on heavyweight cryptographic proofs or distributed trust assumptions, the proposed approach records hash-chained update logs that enable efficient and tamper-evident provenance tracking. This design provides end-to-end accountability across diverse update types and supports step-level verification within standard LLM workflows. By integrating verifiability into the update process itself, the proposed framework bridges the gap between empirical diagnostics and fully accountable model evolution. Importantly, AuditableLLM is intended as a lightweight accountability substrate that is complementary to stronger cryptographic attestations (e.g., PoL/ZK) and certified unlearning protocols: when stronger adversarial guarantees are required, such mechanisms can be layered on top of the same logged update transactions.

Table 1 shows that these directions target different certified objects under different threat models, and therefore are not directly comparable under a single experimental protocol. In particular, PoL/ZK methods aim to attest training correctness under strong adversaries, whereas certified unlearning focuses on forgetting correctness or bounded influence, and blockchain-style systems provide replicated immutability via consensus. AuditableLLM instead targets low-friction, tamper-evident update traceability for frequent PEFT, continual learning, and compliance-driven unlearning operations under an honest-but-curious provider, and can serve as a common logging substrate onto which stronger attestations can be layered when required.

Table 1. Comparison with representative assurance mechanisms across threat and trust model, certified object, and deployment requirements.

Approach	**Threat and Trust Model**	**Certified Object**	**Typical Deployment Requirement**
PoL/PoT, ZK proofs of training [15,16,21]	Strong/fully malicious provider	Training executed as claimed (strong attestation of training provenance)	Requires proof-generation hooks, access to training transcripts/commitments, and verification infrastructure
Certified/verifiable unlearning, unlearning auditing [27,31–33]	Typically honest-but-curious to stronger settings (protocol-dependent)	Forgetting correctness/bounded influence of removed data	Often requires additional training steps, influence bounding, or extra evaluation rounds for certification
Blockchain-based audit logs [22]	Distributed trust/multi-party consensus	Tamper-resistant replicated history (ledger immutability)	Requires consensus/replication network and ledger maintenance in addition to local logging
Provenance/governance logging (no crypto chaining) [4,23]	Benign logging assumption	Operational lineage and documentation (non-cryptographic)	Standard logging pipeline; no cryptographic integrity binding
AuditableLLM (this work)	Honest-but-curious provider; verifiable third-party auditing	Tamper-evident update-history integrity + step-level traceability for adaptation/unlearning	Requires only local hash-chained logging and verification; no retraining proofs or consensus network

3. System Model and Design Objectives

To support verifiable and regulation-ready model evolution, this section formalizes the system foundations of the proposed AuditableLLM framework. It first outlines the system model, describing the actors and lifecycle of auditable model updates. It then introduces the conceptual architecture that links adaptation, audit, and verification into a unified accountability pipeline. Finally, it clarifies the design objectives that shape the framework's requirements for transparency, efficiency, and trustworthy verification.

3.1. System Model Overview

The proposed AuditableLLM framework is modeled as a compliance-aware and verifiable framework in which an LLM evolves through a series of update operations, such as training, fine-tuning, continual learning, correction, or unlearning, implemented via parameter-efficient or modular mechanisms. Each operation modifies the model's adaptable parameters while preserving the base model's integrity, and simultaneously produces auditable evidence that can later be verified by independent parties.

As illustrated in Figure 1, the system involves three principal actors. First, data contributors or requesters provide training data or issue modification and deletion requests under regulatory provisions such as the GDPR's right to erasure [1]. In practical deployments, these entities can be instantiated as clients or tenants (e.g., in multi-tenant or federated settings) and may be benign or adversarial. Accordingly, each update request can be treated as a client-scoped transaction whose provenance is bound to a unique, stable client/tenant identifier and committed into the audit log, enabling post-hoc attribution, investigation, and rollback when suspicious or non-compliant updates are identified. Depending on deployment and audit-log visibility, this identifier can be stored in clear for closed settings or as a privacy-preserving token with an access-controlled resolution mechanism, without weakening client-scoped accountability. Their update requests, either for training or unlearning, are submitted to the model provider.

The model provider is responsible for maintaining the base model, executing these updates within the Auditable Framework, and generating corresponding audit records. Each update produces both an updated model state and cryptographically protected audit logs that capture operation metadata and integrity hashes.

Finally, auditors or regulators independently verify these logs without accessing proprietary parameters or raw data. This end-to-end process forms a transparent and tamper-evident audit trail, ensuring accountability, verifiability, and regulatory readiness across the entire model lifecycle [31,33].

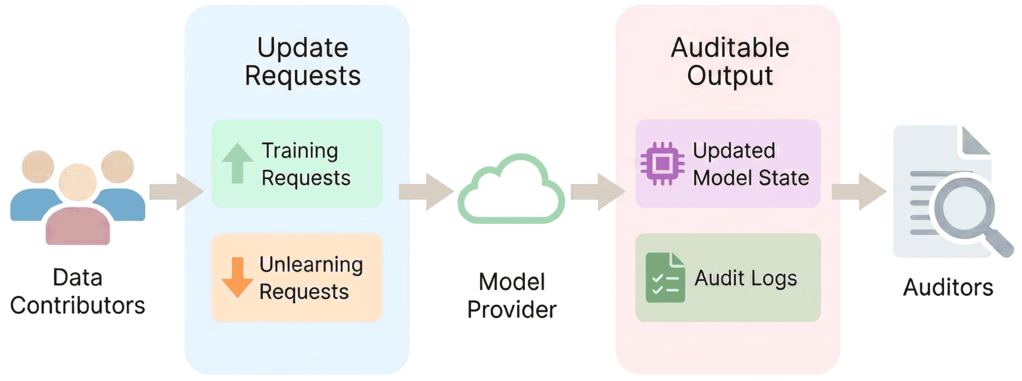

Figure 1. Layered conceptual architecture of the AuditableLLM framework.

Although the framework is model-agnostic and compatible with diverse adaptation algorithms, the experiments instantiate it with a LoRA-based parameter-efficient unlearning baseline. The baseline applies approximate influence-based parameter correction to the adapter parameters, following the unified parameter-efficient unlearning algorithm of [34], to demonstrate generality, efficiency, and reproducibility.

3.2. Conceptual Components

The proposed AuditableLLM is designed as a general and compliance-aware audit framework for LLMs. It consists of three interacting components: data and adaptation, audit, and verification and compliance layers, which together define the logical architecture of an auditable and regulation-ready LLM system. These components collectively form the conceptual foundation of the framework, while their operational realization is detailed in Section 4. Each component interacts with the others to establish a closed accountability loop, from model updates, to audit recording, to independent verification. Figure 2 illustrates the layered architecture of the proposed AuditableLLM framework and the interactions among its three components.

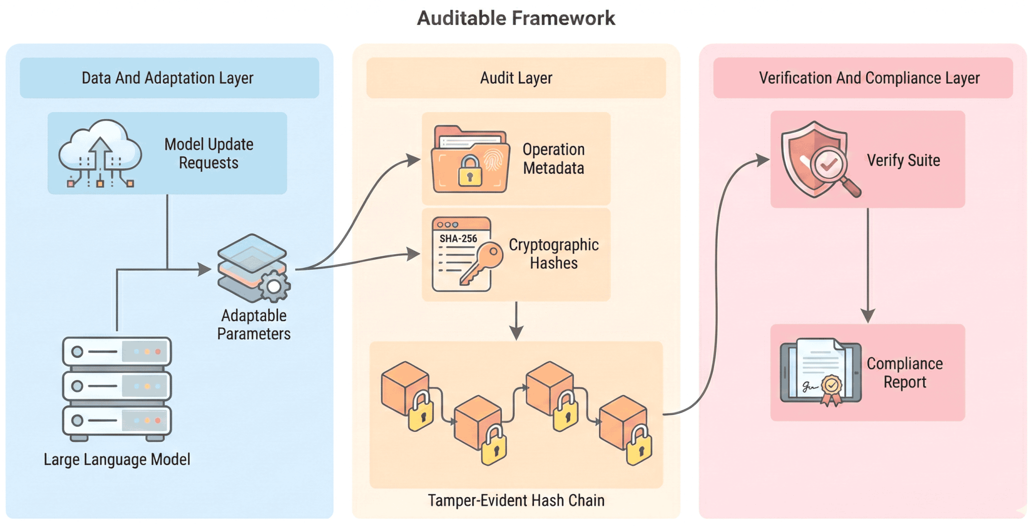

Figure 2. Layered conceptual architecture of the proposed AuditableLLM framework.

3.2.1. Data and Adaptation Layer

At the foundation lies a dynamic process of parameter-efficient model adaptation. A pretrained LLM serves as the immutable backbone, while lightweight modules such as LoRA, QLoRA, or prefix adapters enable continual updates, fine-tuning, or compliance-driven modifications. Depending on operational or regulatory requirements, the data used for model updates may undergo lifecycle management, including correction, replacement, or removal. Every model update, whether for training, correction, or maintenance, is therefore treated as an auditable event that modifies only the adaptable parameters while preserving the integrity of the base model. Machine unlearning is regarded as a compliance-oriented instance within this broader LLM adaptation paradigm [35,36].

3.2.2. Audit Layer

The audit layer serves as the cryptographic backbone of AuditableLLM, maintaining a tamper-evident and append-only record of all adaptation events. Rather than storing raw data or full model parameters, it records pseudonymized metadata and cryptographic digests that summarize each update operation. These entries are chronologically linked to create a lightweight hash-chain structure, so that any alteration, omission, or replay attempt can be externally detected. By relying on standard cryptographic primitives (e.g., SHA-256, Merkle-style linking) instead of heavy zero-knowledge proofs, the audit layer achieves a practical balance between verifiability and computational efficiency [22,31]. Its modular architecture ensures compatibility with various LLM adaptation methods and seamless integration with existing compliance workflows or version-control systems.

3.2.3. Verification and Compliance Layer

The verification layer establishes trust and accountability between the model provider and external auditors. Assuming an honest-but-curious model provider, who correctly executes updates but must demonstrate compliance, verification operates on two complementary levels. Structural verification ensures the cryptographic integrity and chronological ordering of audit records, while behavioral verification evaluates whether the LLM's performance and behavior remain consistent with intended learning objectives. Together, these mechanisms allow independent auditors to confirm the authenticity, reliability, and compliance of all recorded operations [32,37].

In summary, these three components jointly define the conceptual foundation of the proposed framework: the data and adaptation layer produces observable parameter updates, the audit layer records them in a verifiable and tamper-evident manner, and the verification layer attests them through independent validation. This definition provides the basis for the operational methodology and system implementation described in Section 4.

3.3. Design Objectives

Building upon the conceptual architecture outlined above, AuditableLLM is designed to ensure trustworthy, transparent, and regulation-ready model adaptation across the full lifecycle of LLMs. Unlike traditional unlearning-focused frameworks, it generalizes auditability to all forms of model evolution, including training, fine-tuning, continual learning, correction, and unlearning, ensuring that every update can be verified, traced, and audited in a consistent and trustworthy manner.

The framework is guided by three overarching objectives:

(i) System-Level Accountability. Every model update, regardless of its type, granularity, or intent, must produce a verifiable and tamper-evident record. This objective aligns with emerging governance frameworks such as the GDPR and the EU AI Act, which emphasize demonstrable traceability, provenance tracking, and lifecycle accountability for AI systems deployed in regulated contexts.

(ii) Technical Efficiency and Modularity. The framework must remain lightweight, parameter-efficient, and model-agnostic, supporting diverse adaptation paradigms while introducing minimal computational or storage overhead. By decoupling the auditing mechanism from the underlying learning algorithm, AuditableLLM enables flexible deployment across heterogeneous infrastructures such as MLaaS, FL, and on-premise compliance environments [8,9].

(iii) Verifiability and Behavioral Reliability. Verification operates on multiple levels: (1) cryptographic integrity checks ensure audit-chain correctness; (2) parametric validation confirms consistency between successive model updates; and (3) behavioral diagnostics evaluate whether the model's outputs remain faithful, safe, and compliant after adaptation. Together, these mechanisms guarantee that both the process and outcomes of model evolution are externally auditable and reproducible [32,37].

Overall, these design principles establish a foundation for building scalable, efficient, and verifiable audit mechanisms applicable to any model update process. They define how the proposed system should function to ensure integrity, transparency, and accountability, setting the stage for the operational methodology detailed in Section 4.

4. Methodology and System Design

This section presents the methodology and system design of the proposed AuditableLLM framework, translating the system model and design principles into an operational, verifiable adaptation pipeline. A unified mechanism for model updates is established, a tamper-evident audit layer is constructed, and a verification and compliance suite is developed to ensure trustworthy and regulation-ready model evolution. Together, these components form a coherent methodology for achieving accountability, efficiency, and verifiability across the full lifecycle of LLM adaptation.

4.1. Methodological Rationale and Pipeline

This section establishes the methodological foundation of the proposed AuditableLLM framework and instantiates it as a unified, verifiable adaptation pipeline for LLMs.

At its core, AuditableLLM integrates model evolution into a closed compliance loop. Any operation—including training, fine-tuning, continual learning, correction, or unlearning—is modeled as a standardized update transaction that passes through three stages: (1) Update Execution, where the provider applies a parameter-efficient or modular update and records the associated metadata; (2) Audit Logging, where a cryptographic digest of the operation is committed to an append-only audit chain; and (3) Verification and Certification, where internal or external auditors validate the structural integrity and behavioral reliability of the resulting model. This unified abstraction places all adaptation behaviors under a single auditable contract and enables consistent verification logic for both learning and unlearning [22,31,33].

Following parameter-efficient and modular adaptation in LLMs [8,9], the framework ensures that model updates remain lightweight, auditable, and reproducible, linking advances in efficient fine-tuning with verifiable machine learning [1,4]. Subsequent sections detail the unified update mechanism, the audit layer, the verification and compliance suite, and the overall security and threat model that together operationalize accountability and verifiability in AuditableLLM.

4.2. Unified Update Mechanism

The first operational module of the proposed AuditableLLM framework is a unified update mechanism that standardizes all forms of model adaptation, including training, fine-tuning, continual learning, correction, and unlearning, under a single verifiable abstraction. This design provides a consistent mathematical and procedural interface for integrating parameter-efficient updates with auditability and compliance assurance.

4.2.1. General Formulation

Let $\mathcal{M}_0$ denote the immutable base model and $\boldsymbol{\theta}_t$ the parameter-efficient component (e.g., LoRA adapter, prefix embedding, or other modular head) at update step t. Each model update operation U_t transforms the current system state $(\mathcal{M}_0, \boldsymbol{\theta}_{t-1})$ into a new configuration $(\mathcal{M}_0, \boldsymbol{\theta}_t)$ via a generalized update operator $\mathcal{F}$:

$$\boldsymbol{\theta}_t = \mathcal{F}(U_t; \boldsymbol{\theta}_{t-1}, \mathcal{D}_t),$$

where $\mathcal{D}_t$ represents the data subset associated with the operation (e.g., task data, correction samples, or deletion requests). This abstraction unifies the semantics of different adaptation modes by treating each as an instantiation of the same functional interface $\mathcal{F}$.

Depending on context, $\mathcal{F}$ may correspond to gradient-based optimization, influence-guided parameter correction, or selective weight reconfiguration. For instance, in compliance-oriented scenarios, U_t may represent a deletion update that counteracts the influence of a forget set $\mathcal{D}_{\text{forget}}$, while in performance-oriented scenarios it may denote a conventional fine-tuning or calibration step. In all cases, the resulting parameter delta $\Delta\boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$ forms the atomic evidence unit to be recorded in the audit layer.

4.2.2. Parameter-Efficient Realization

To ensure scalability within LLM adaptation, the operator $\mathcal{F}$ is instantiated through PEFT paradigms such as LoRA, QLoRA, or prefix-tuning [8,9]. Rather than retraining the entire base model $\mathcal{M}_0$, these methods apply localized updates to $\boldsymbol{\theta}_t$, typically accounting for less than 1% of the total parameters. This modularity substantially reduces computational cost and storage footprint, while enabling precise auditability: each parameter delta $\Delta\boldsymbol{\theta}_t$ can be independently hashed, recorded, and verified by the audit layer.

As one representative case, LoRA-based modules decompose the trainable component into a pair of low-rank matrices $(\mathbf{A}_t, \mathbf{B}_t)$ such that

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \mathbf{B}_t\mathbf{A}_t.$$

The resulting incremental updates $\Delta\boldsymbol{\theta}_t = \mathbf{B}_t\mathbf{A}_t$ represent verifiable adaptation events that are subsequently captured by the audit mechanism. The same abstraction applies to other PEFT variants, ensuring consistency and interoperability across different LLM adaptation workflows.

4.2.3. Audit Integration

Each update transaction U_t emits two synchronized outputs: the updated parameter module $\boldsymbol{\theta}_t$ and the corresponding audit metadata $\boldsymbol{\mu}_t$. The metadata includes operation identifiers, timestamped commitments, and cryptographic hashes of $\Delta\boldsymbol{\theta}_t$, which are immediately committed to the audit chain. This ensures that every update, whether it introduces new knowledge or removes prior information, creates a tamper-evident record linking algorithmic execution to verifiable evidence.

By abstracting all forms of adaptation through a unified interface and coupling them with real-time audit generation, the proposed mechanism transforms model updates into cryptographically attestable events. This foundation enables the seamless integration of efficiency, accountability, and verifiability across diverse operational contexts, forming the backbone of the AuditableLLM pipeline.

4.3. Audit Layer Construction

The audit layer forms the cryptographic backbone of the proposed AuditableLLM framework. Its role is to transform every model update into a tamper-evident and externally verifiable record, ensuring that the history of model evolution remains transparent, traceable, and compliant. Unlike conventional version-control or logging systems, this layer emphasizes verifiable accountability rather than mere record-keeping, as each update transaction is cryptographically committed, timestamped, and chained to its predecessor to form an immutable lineage of model states.

4.3.1. Audit Record Structure

For each update operation U_t, the model provider generates an audit record $\mathbf{R}_t$ containing a compact yet expressive set of metadata:

$$\mathbf{R}_t = \{\mathrm{ID}_t, \boldsymbol{\mu}_t, \mathbf{h}_{\Delta\boldsymbol{\theta}_t}, \mathbf{h}_{t-1}, \mathrm{timestamp}_t\}.$$

Here, ID_t denotes a unique, stable operation identifier, and $\boldsymbol{\mu}_t$ represents operation metadata (e.g., update type, data tag, compliance reason). In multi-tenant or federated deployments, ID_t can be instantiated as a tuple that includes a tenant-scoped client identifier, e.g., $\mathrm{ID}_t = (\mathrm{CID}_t, \mathrm{op}_t)$, enabling client-scoped attribution of update transactions. Depending on deployment and audit-log visibility, CID_t can be stored in clear in closed settings or represented as a privacy-preserving token that remains resolvable by the provider/authorized auditors under access control, without weakening accountability. Optionally, $\boldsymbol{\mu}_t$ may include a digest of the client-submitted request or batch (e.g., $\mathbf{h}_{\mathcal{D}_t} = H(\mathcal{D}_t)$) and a client-side signature over the request summary to provide non-repudiation at the logging boundary. The term $\mathbf{h}_{\Delta\boldsymbol{\theta}_t} = H(\Delta\boldsymbol{\theta}_t)$ is the cryptographic hash of the parameter delta, and $\mathbf{h}_{t-1}$ is the hash of the previous record. Finally, in the implementation, $H(\cdot)$ is instantiated as the SHA-256 hash function, a widely deployed 256-bit cryptographic primitive. The hash-chain construction itself does not rely on any additional pseudo-random generator: its tamper-evidence guarantees follow from the collision and (second-)preimage resistance of SHA-256. Any random identifiers or nonces that may appear in the metadata $\boldsymbol{\mu}_t$ are drawn from the operating system's cryptographically secure random source, but they are not required for the security of the chaining mechanism. The inclusion of $\mathbf{h}_{t-1}$ ensures that all audit records are interlinked, forming a chronological hash chain $\mathcal{L} = \{\mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_T\}$.

Each record thus binds three dimensions of verifiability: (i) temporal integrity, ensured by chronological linkage; (ii) semantic accountability, ensured by metadata commitments; and (iii) structural traceability, ensured by cryptographic hashing of model deltas.

4.3.2. Hash-Chain Construction

The audit layer implements an incremental hash-chain mechanism that sequentially links all update events. As illustrated in Figure 3, each audit record $\mathbf{R}_t$ encapsulates operation metadata $(\boldsymbol{\mu}_t)$, the parameter digest $(\mathbf{h}_{\Delta\theta_t})$, the previous chain hash $(\mathbf{h}_{t-1})$, and a timestamp (t). These components are concatenated and hashed to produce the new chain state $\mathbf{h}_t$, ensuring that any modification or omission invalidates all subsequent records.

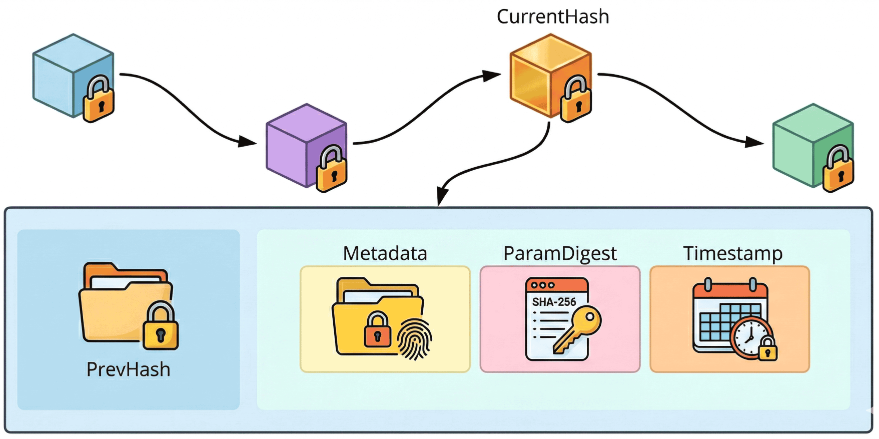

Figure 3. Conceptual diagram of the hash-chain construction used in the AuditableLLM audit layer.

Formally, given the previous hash $\mathbf{h}_{t-1}$ and a new record $\mathbf{R}_t$, the current chain hash $\mathbf{h}_t$ is computed as:

$$\mathbf{h}_t = H\big(\mathbf{h}_{t-1} \,\|\, H(\boldsymbol{\mu}_t \,\|\, \mathbf{h}_{\Delta\theta_t} \,\|\, t)\big),$$

where $\|$ denotes concatenation. In all experiments, H is set to SHA-256. Under standard cryptographic assumptions, forging a different audit history that yields the same final chain head $\mathbf{h}_T$ would require either finding a collision or a second preimage for SHA-256. Given the 256-bit output size and the scale of audit logs considered in the evaluated setting (up to 10^4–10^5 records), the probability that any computationally bounded adversary can mount such an attack is negligible. Consequently, SHA-256 provides more than sufficient security for the goal of tamper-evident audit logging without incurring additional overhead from heavier primitives or consensus protocols. This lightweight design maintains cryptographic immutability similar to blockchain ledgers, yet avoids consensus overhead and remains self-contained within the model provider's infrastructure.

Algorithm 1 summarizes the operational process for constructing the tamper-evident audit chain. Each model update is committed with its metadata and parameter digest, extending the verifiable history of model evolution.

Algorithm 1: Tamper-Evident Audit Chain Construction for AuditableLLM

```
Input: Previous hash $\mathbf{h}_{t-1}$, operation metadata $\boldsymbol{\mu}_t$, parameter delta $\Delta\boldsymbol{\theta}_t$,
       timestamp $t$
Output: Updated chain state $\mathbf{h}_t$
1 Compute parameter digest $\mathbf{h}_{\Delta\theta_t} \leftarrow H(\Delta\boldsymbol{\theta}_t)$
2 Assemble record $\mathbf{R}_t = (\boldsymbol{\mu}_t, \mathbf{h}_{\Delta\theta_t}, \mathbf{h}_{t-1}, t)$
3 Compute new chain hash $\mathbf{h}_t \leftarrow H(\mathbf{h}_{t-1} \| H(\boldsymbol{\mu}_t \| \mathbf{h}_{\Delta\theta_t} \| t))$
4 Store $\mathbf{R}_t$ and $\mathbf{h}_t$ in the append-only audit log $\mathcal{L}$
5 return $\mathbf{h}_t$
```

4.3.3. Provider-Side Procedure

For each update operation U_t, the provider constructs a new audit record and updates the chain state in a fixed sequence of steps. Given the previous chain head $\mathbf{h}_{t-1}$ (or a designated genesis value for $\mathbf{R}_0$), the operation metadata $\boldsymbol{\mu}_t$, the parameter delta $\Delta\boldsymbol{\theta}_t$, and a timestamp t, the provider first computes the parameter digest $\mathbf{h}_{\Delta\boldsymbol{\theta}_t} = H(\Delta\boldsymbol{\theta}_t)$. It then assembles the record payload $(\boldsymbol{\mu}_t, \mathbf{h}_{\Delta\boldsymbol{\theta}_t}, \mathbf{h}_{t-1}, t)$. Finally, the new chain head $\mathbf{h}_t$ is derived by hashing the previous head together with the payload hash. The resulting record $\mathbf{R}_t$ and updated head $\mathbf{h}_t$ are appended to an append-only storage backend. Algorithm 1 summarizes this per-update procedure.

4.3.4. Design Considerations

The audit layer is designed for modular integration and low overhead. It does not store raw data, gradients, or model weights, but only compact digests and pseudonymized metadata. As a result, storage costs grow linearly with the number of updates rather than model size, and the entire verification process can be executed efficiently even for large-scale LLMs. Furthermore, this layer is compatible with privacy-preserving techniques such as differential privacy and secure multiparty computation, allowing audit proofs to be published or verified without disclosing proprietary or sensitive information [4,22]. In privacy-preserving fine-tuning or federated learning settings, the audit metadata can additionally record declared DP-related configurations (e.g., clipping norms, noise scales, and cumulative privacy budgets) for each update or round. This enables auditors to check the internal consistency of the claimed privacy parameters over the lifecycle of training and unlearning. However, the actual noise generation and application remain encapsulated inside a trusted execution environment. A systematic, formal quantification of potential information leakage from parameter digests or audit metadata, and a tighter integration of the audit layer with certified privacy-preserving training or unlearning methods and protocol-level guarantees that DP mechanisms were executed as configured, are promising directions for future work.

In summary, the audit layer converts every algorithmic operation into an immutable, verifiable event. It serves as the cryptographic foundation that links model adaptation with compliance assurance, providing regulators and auditors with tamper-evident evidence of each model evolution step. This verifiable audit trail forms the basis for the integrity and accountability guarantees of the AuditableLLM framework.

4.3.5. Log Storage and Retention

AuditableLLM treats the storage backend for audit records as a pluggable component. In the prototype implementation, hash-chained logs are persisted as append-only records on provider-managed storage. The abstraction is nonetheless compatible with both internal databases (e.g., append-only tables or key-value stores) and external write-once or versioned object stores that offer stronger tamper-resistance guarantees. The framework does not mandate a specific retention window. Instead, deployments are expected to choose log retention policies that align with applicable regulatory and organizational requirements, such as retaining full histories for high-risk systems or enforcing time-bounded retention for low-risk deployments. When logs are pruned or archived, new checkpoints of the hash chain head can be anchored (e.g., via periodic digests) so that the integrity of the remaining prefix remains verifiable.

4.4. Verification and Compliance Layer

The verification and compliance layer constitutes the trust and assurance component of the AuditableLLM framework. While the audit layer guarantees the immutability of recorded updates, this layer ensures that all updates are both cryptographically verifiable and behaviorally faithful, bridging the technical and empirical dimensions of accountability.

4.4.1. Structural Verification

This submodule focuses on the cryptographic validation of audit integrity and parameter consistency. Given an updated model $\mathcal{M}_t$ and its audit segment $\{\mathbf{R}_{t-k}, \ldots, \mathbf{R}_t\}$, the verifier performs:

1. Hash-Chain Validation: Ensures that audit records form an unbroken chronological chain, confirming that no record has been inserted, removed, or modified.
2. Parameter Commitment Verification: Recomputes digest commitments $H(\Delta\boldsymbol{\theta}_t)$ from model parameters and checks equality with logged commitments, verifying that the recorded deltas match actual model updates.

These steps establish structural integrity and parametric consistency, the cryptographic foundation of verifiable model accountability.

4.4.2. Computational Complexity

Let T denote the number of recorded update events and C_H the cost of evaluating the hash function once. To make comparisons explicit across assurance mechanisms, the following notation is additionally used: C_{scan} for the constant-time per-artifact overhead of parsing/serialization and I/O-bound checks (made explicit for log-style schemes where verification is a linear scan over T records; analogous costs exist in other paradigms but are typically dominated by their primary primitives); C_{sig} for verifying one digital signature; C_{CHF} for one chameleon-hash/skip-link consistency check when applicable; $C_{\text{step}}(|W|)$ for the cost of one training/update step for parameter size $|W|$; and $V_{\text{SNARK}}(n, |io|)$ for the verifier cost of validating a SNARK proof for a circuit/R1CS of size n with public I/O size $|io|$. Here, n is induced by the computation being proved (e.g., the specific training/unlearning procedure and its model/state encoding), while the verifier cost is typically succinct (often sub-linear in n and in some constructions dominated by $|io|$). For replay-based proof-of-learning style verification, let E denote the number of epochs, Q the number of verifier queries per epoch, and k the number of steps per queried segment. For federated logging schemes, let N_c denote the number of client contributions whose signatures may be verified per update.

On the provider side, creating and appending an audit record for a single update requires a constant number of hash evaluations together with lightweight serialization and storage I/O. As a result, the total logging overhead across T updates scales as $O(T\,(C_H + C_{\text{scan}}))$ in time and $O(T)$ in storage, independent of the size of the underlying model. On the auditor side, the structural verification procedure (Algorithm 2) processes each record exactly once to recompute payload hashes and chain heads, and, when model states are available, it can additionally recompute parameter digests. This yields a worst-case and typical verification complexity of $O(T\,(C_H + C_{\text{scan}}))$ time and $O(1)$ auxiliary memory beyond the log itself, with early termination if a mismatch is encountered. In other words, the cost of a full integrity check grows linearly with the length of the audit log. Table 2 summarizes the asymptotic verifier-side upper-bound cost of AuditableLLM in comparison with representative proof-of-learning, certified/unlearning-auditing, blockchain-based, and provenance-logging approaches, using a unified notation that makes the dominant per-update primitives (training-step replay, proof verification, signature verification, and hash/scan operations) explicit.

Table 2. Unified asymptotic upper-bound verifier cost of representative assurance mechanisms.

Approach	**Verification Cost & Overhead**
Proof-of-learning/proof-of-training (replay-based) [15,16]	$O\big(E\,Q\,k\,C_{\text{step}}(\lvert W\rvert)\big)$
ZK proofs of training/verifiable computation (SNARK-based) [21]	$O\big(V_{\text{SNARK}}(n, \lvert io\rvert)\big)$
Verifiable/certified machine unlearning and unlearning auditing [27,31,32]	$O\big(T\,V_{\text{cert}} + R_u\,C_{\text{eval}}\big)$
Blockchain-based federated logging (e.g., VFChain) [22]	$O\big(T\,(C_H + C_{\text{scan}} + (N_c+1)\,C_{\text{sig}} + C_{\text{CHF}})\big)$
Standard provenance logging/governance (no cryptographic chaining) [4,23]	$O\big(T\,C_{\text{scan}}\big)$
AuditableLLM (this work)	$O\big(T\,(C_H + C_{\text{scan}})\big)$

Algorithm 2: Auditor-side verification of audit log and hash chain

```
Input: Sequence of audit records $\{\mathbf{R}_1, \ldots, \mathbf{R}_T\}$, optional model states
       $\{\mathcal{M}_0, \ldots, \mathcal{M}_T\}$
Output: Verdict ACCEPT or REJECT
1  Initialize expected head $\hat{\mathbf{h}}_0$ (e.g., genesis value or trusted checkpoint)
2  for $t \leftarrow 1$ to $T$ do
3      Parse $\mathbf{R}_t$ into $(\mathrm{ID}_t, \boldsymbol{\mu}_t, \mathbf{h}_{\Delta\boldsymbol{\theta}_t}, \mathbf{h}^{\log}_{t-1}, \mathrm{timestamp}_t)$
4      if $\mathbf{h}^{log}_{t-1} \neq \hat{\mathbf{h}}_{t-1}$ then
5          return REJECT                                  // broken chain link
6      if model states $\mathcal{M}_{t-1}, \mathcal{M}_t$ are available then
7          Recompute $\Delta\boldsymbol{\theta}_t$ from $\mathcal{M}_{t-1} \to \mathcal{M}_t$
8          if $H(\Delta\boldsymbol{\theta}_t) \neq \mathbf{h}_{\Delta\boldsymbol{\theta}_t}$ then
9              return REJECT                         // parameter digest mismatch
10     Compute payload hash $c_t \leftarrow H(\boldsymbol{\mu}_t \,\|\, \mathbf{h}_{\Delta\boldsymbol{\theta}_t} \,\|\, \mathrm{timestamp}_t)$
11     Update chain head $\hat{\mathbf{h}}_t \leftarrow H(\hat{\mathbf{h}}_{t-1} \,\|\, c_t)$
12 return ACCEPT                  // log and chain are structurally consistent
```

4.4.3. Verification Procedure

Algorithm 2 makes the structural verification steps explicit from the auditor's perspective. The auditor processes the audit records sequentially, checking that each record's stored previous head matches the recomputed chain head. At each step, the record hash is re-derived from the metadata, parameter digest, and timestamp, and the expected chain head is updated accordingly. When intermediate model states are available, the auditor can additionally reconstruct the parameter delta at each step and compare its digest to the logged value. Any mismatch in these checks causes the audit log to be rejected, whereas a fully consistent pass yields an acceptance decision.

4.4.4. Behavioral Verification

Beyond structural checks, behavioral verification ensures that the LLM's functional behavior aligns with declared learning or unlearning objectives. This involves empirical testing on non-sensitive validation subsets to assess: (i) knowledge retention or correction stability after fine-tuning, and (ii) forgetting effectiveness or privacy preservation after unlearning. Such assessments may employ task-specific evaluation probes or privacy-risk metrics, but the general mechanism remains model-agnostic and data-minimized. Specific metric definitions are provided in Section 5.

4.4.5. Compliance Certification

When both structural and behavioral verifications succeed, the auditor issues a compliance certificate $\mathcal{C}_t$ summarizing verified outcomes for update U_t. Each certificate contains: (i) the audit chain checkpoint $\mathbf{h}_t$; (ii) verification timestamp and auditor ID; and (iii) signed attestations of both structural and behavioral compliance. These certificates provide tamper-evident, regulator-ready evidence of accountability.

4.4.6. Operational Flexibility

The layer supports two verification modes: partial verification (structural checks only) for lightweight audits, and full verification (structural + behavioral) for formal regulatory review. All computations depend only on cryptographic digests and non-sensitive validation data, ensuring privacy-preserving operation across MLaaS, federated, or on-premise deployments.

4.5. Threat Model and Security Assurance

Building upon the accountability guarantees established in previous sections, this section extends the analysis from verifiable operations to active security assurance. The objective is to ensure that the proposed AuditableLLM framework remains verifiable, tamper-evident, and trustworthy throughout the entire model adaptation lifecycle, even under partially untrusted environments.

4.5.1. Adversary Model

Throughout this work, the focus is on an honest-but-curious model provider. The provider faithfully executes the prescribed training and unlearning operations, but may later attempt to conceal, omit, or manipulate audit information to hide non-compliant updates. Potential adversarial behaviors include: (i) tampering, modifying or rewriting previous audit entries; (ii) omission, selectively removing records to obscure certain operations; (iii) replay, reusing outdated logs to falsify system state; and (iv) leakage, unintentionally retaining or exposing sensitive data from previous updates. Auditors are assumed to be independent and non-colluding, but do not have access to proprietary model parameters or raw data. This is distinguished from a stronger, fully malicious provider who could, in principle, train a model entirely offline and subsequently fabricate an entirely synthetic but structurally valid audit log that is consistent with the final model parameters, and then present this log as if it were the true history. This paper does not attempt to cryptographically exclude such "fake history" scenarios; instead, once the provider instruments its pipeline with the audit layer, the hash-chain-backed log ensures that any ex post modification, deletion, or reordering of the resulting history becomes efficiently detectable.

4.5.2. Malicious Clients/Contributors

Beyond provider-side misbehavior, adversarial clients are also considered who may (i) issue strategically crafted correction and deletion requests and subsequently deny or repudiate the submitted requests or their effects, or (ii) submit poisoned or backdoored training samples. AuditableLLM provides a verifiable client-scoped accountability substrate by binding each update transaction to a unique, stable tenant-scoped client identifier (stored in clear in closed deployments or represented as a resolvable privacy-preserving token under access control), and by committing request digests, declared data tags, and compliance reasons into the audit log. This design prevents the denial or repudiation component of (i) at the logging boundary (via request digests and optional client-side signatures) and constrains abuse of (i) crafted correction and deletion requests by enabling authorization checks and producing tamper-evident evidence for post-hoc verification, dispute resolution, and rollback to a known-good checkpoint when suspicious requests are detected. For (ii) poisoning/backdoor submissions, AuditableLLM ensures fine-grained attribution and traceability across updates; complementary defenses (e.g., admission control, filtering, and robust aggregation) can be plugged into the update execution stage, while the audit layer records the defense configuration and the resulting accepted or rejected client updates for later verification.

4.5.3. Detection and Defense

AuditableLLM integrates multiple mechanisms to detect and contain such adversarial behaviors. Tampering and omission are exposed through hash-chain recomputation, where any change or reordering of records produces a digest mismatch. Parameter inconsistency, in which the committed Δ_t hash does not match the actual model state, is detected through independent recomputation and cross-validation. Incomplete or unverifiable records are identified via missing timestamps, broken hash links, or invalid ordering, ensuring that the audit trail remains continuous and cryptographically verifiable.

4.5.4. Behavioral Anomaly Detection

Beyond structural verification, the framework includes empirical mechanisms for detecting behavioral anomalies across model updates. Independent auditors or monitoring agents may periodically assess the model's externally observable behavior on non-sensitive benchmark subsets, evaluating whether performance deviations exceed expected operational bounds. Significant drifts in accuracy, robustness, or fairness metrics may indicate unrecorded updates, model tampering, or unintended data influence. These behavioral diagnostics complement cryptographic verification, providing additional evidence of compliance and stability across adaptation cycles.

4.5.5. Trust Boundary

A clearly defined trust boundary separates the model provider, responsible for generating audit records and executing updates, from the auditor, who independently validates their integrity and behavioral soundness. The auditor never accesses proprietary model weights or raw data; verification relies solely on cryptographic digests, metadata, and externally observable behaviors. This separation enforces accountability and verifiability without requiring mutual trust or sensitive data disclosure.

4.5.6. Security Assurance

Within the honest-but-curious setting described above, the proposed framework combines structural hash verification with behavioral validation to provide a practical assurance layer that detects both cryptographic and functional deviations. All audit records are internally self-contained and reproducible, so that an auditor can re-derive and check the hash chain without relying on external notarization or special storage infrastructure in this baseline configuration. At the same time, these assurances are conditional on the provider actually instrumenting its training and unlearning pipeline with the audit layer. A fully malicious provider who trains models entirely offline and later fabricates a self-consistent but counterfeit audit log is therefore outside the adversary class that AuditableLLM is designed to handle directly. Strengthening guarantees to this stronger setting requires binding the audit process to external roots of trust, such as trusted time-stamping services, hardware-secured or remotely attested logging, or proof-of-learning style protocols that tie intermediate model states to real executions. Exploring such compositions is beyond the scope of this proof-of-concept study, but AuditableLLM's hash-chain-backed logging layer is intended to serve as a reusable substrate for these heavier attestation mechanisms. Under the stated threat model, this design yields strong, efficiently checkable accountability for each adaptation step and offers a natural foundation on top of which more comprehensive, end-to-end secure compliance solutions can be built.

4.6. Integration and Problem-Solution Alignment

This section concludes the methodology by illustrating how the proposed design realizes the system-level objectives introduced earlier. Each methodological module of AuditableLLM directly addresses a specific challenge identified in the design phase, collectively forming an integrated, verifiable, and regulation-ready adaptation framework. Table 3 summarizes this problem–solution alignment by mapping each design objective to its corresponding challenge and methodological realization.

Table 3. Alignment between design objectives, challenges, and methodological realizations in the AuditableLLM framework.

Design Objective	**Challenge**	**Methodological Realization**
System-Level Accountability	Absence of verifiable lineage for model evolution and data operations.	Hash-chained audit logs with cryptographic digests to ensure tamper-evident, verifiable update history.
Technical Efficiency	High retraining cost and instability in parameter updates.	Parameter-efficient modular fine-tuning (LoRA/QLoRA) enabling lightweight, stable model adaptation.
Verifiability and Reliability	Unverifiable or inconsistent model behavior after adaptation.	Dual-layer verification combining structural hash checks and behavioral consistency evaluation.

Together, these methodological realizations operationalize the theoretical design principles outlined in the system model. By linking efficiency, accountability, and verifiability into a unified framework, AuditableLLM provides a concrete pathway for building compliance-aware and audit-ready LLM systems. This integration closes the methodological loop between conceptual design and practical implementation, forming the foundation for the experimental evaluation presented in Section 5.

5. Experiments

5.1. Experiment Overview

This work designs four complementary experiments (E0, E1, E2, and E3) to comprehensively evaluate the proposed AuditableLLM framework. Each experiment targets a distinct verification layer of the unified audit mechanism:

- E0: Unified Audit Pipeline validates the framework-level stability, integrity, and efficiency of the hash-chained audit layer.
- E1: Auditable Fine-tuning verifies that the audit layer integrates into fine-tuning without affecting model performance or convergence stability.
- E2: Auditable Unlearning examines whether audit-integrated unlearning preserves forgetting efficacy and utility retention relative to the non-audited baseline.
- E3: Membership Inference Robustness evaluates whether unlearned models reduce membership leakage on the forget set to the level of a retrained-from-scratch baseline.

Together, these experiments assess the stability, non-intrusiveness, verifiability, and privacy implications of the framework across the model adaptation lifecycle, from update execution to compliance-oriented data deletion and external verification.

5.2. Experimental Setup

5.2.1. Dataset

All experiments are conducted on the MovieLens-small dataset [38], which contains 100,000 user-movie ratings from 610 users across approximately 9000 movies. Each record is reformulated into an instruction-response pair that asks the model to predict a 1–5 star rating given the user and movie context. This setup provides a lightweight and interpretable benchmark for evaluating auditable model adaptation processes, including personalized fine-tuning, continual learning, and compliance-related updates. This dataset is used to validate the audit-layer properties (e.g., tamper-evidence, step-level traceability, and per-update overhead) under a controlled adaptation/unlearning workflow.

The dataset is chronologically partitioned into 80% training, 10% validation, and 10% testing. Within the training partition, a 10% subset is randomly designated as compliance-sensitive data, simulating deletion, correction, or restricted-use cases. The remaining data form the standard pool for general model updates. For influence-based adaptation experiments, a smaller batch subset $\mathcal{D}_{\text{mini}}$ is sampled from the training data to efficiently approximate gradient and influence computations. Accordingly, task-level generalization to larger and more diverse instruction corpora is treated as a limitation of the current empirical study. This limitation, however, does not alter the applicability of the auditing mechanism itself: the audit layer attaches to update transactions and its runtime cost is driven primarily by the number of logged update events and hash evaluations, rather than by the dataset size directly.

5.2.2. Model

The Llama-3.2-1B-Instruct model [39] is fine-tuned using parameter-efficient adapters implemented via the LoRA configuration [8]. Although AuditableLLM is model-agnostic and compatible with various PEFT methods (e.g., QLoRA, prefix-tuning, adapter fusion), LoRA is chosen for clarity and reproducibility, as it provides a modular and transparent structure that aligns naturally with the audit instrumentation. Each model update, whether for fine-tuning, correction, or unlearning, is automatically logged by the audit layer, which records operation metadata and cryptographic digests in real time.

5.2.3. Hardware

All experiments are conducted on a single workstation equipped with an NVIDIA RTX 4060 GPU (8 GB VRAM), an AMD Ryzen 7 5800H CPU, and 32 GB of system memory. This configuration represents a practical mid-range environment, demonstrating that the proposed audit layer can operate efficiently even on commodity hardware with negligible performance degradation. Timing, throughput, and audit-overhead metrics are averaged over five independent runs to ensure statistical robustness and reliability.

5.2.4. Training Procedure

All experiments are implemented in PyTorch (https://pytorch.org/) using the Hugging Face `transformers` and `peft` libraries. A standard PEFT setup is used, where the base LLM is kept frozen and only LoRA adapter parameters are updated during training. As a proof-of-concept, all runs are executed as single-process, single-device jobs on the workstation described above. No data-, tensor-, or pipeline-parallel distributed training is employed, which is sufficient for the 1B-parameter model considered here.

5.2.5. Membership Inference Protocol

For the MIA-based evaluation in Experiment E3, a standard loss-based membership inference attack is followed. A binary classification task is constructed in which samples from the forget set $\mathcal{D}_{\text{forget}}$ are treated as members, and an equal-sized subset of held-out test samples are treated as non-members. For a given model M and a labeled example (x, y), the adversary observes the per-example training loss $\ell_M(x, y)$ (e.g., cross-entropy or regression loss) and uses its negative value $s(x, y) = -\ell_M(x, y)$ as a membership score. A threshold classifier $A(x, y) = \mathbb{K}[s(x, y) \geq \tau]$ is calibrated on a disjoint calibration split and evaluated on an independent test split. The ROC-AUC of this attack, its accuracy, and the membership advantage $\text{Adv} = \text{TPR} - \text{FPR}$ on $\mathcal{D}_{\text{forget}}$ are reported, all on a $[0, 1]$ scale. Values close to 0.5 (for AUC and accuracy) and 0 (for advantage) indicate that the attack is nearly random.

5.2.6. Reproducibility Details

To improve reproducibility across the entire experimental section (E0–E3), all random seeds (data split, sampling, and training) are fixed and a consistent implementation configuration is used throughout all runs. Unless stated otherwise, AdamW is used with learning rate 1×10^{-4}, weight decay 1×10^{-5}, gradient clipping at 1.0, cosine learning-rate scheduling with a 100-step warmup, per-device batch size 4 with gradient accumulation of 8, and mixed-precision (bf16) training. The maximum sequence length is set to 256 tokens, and deterministic execution is enabled with a fixed seed (1234). For LoRA, rank $r = 8$, $\alpha = 32$, dropout 0.1, and target modules {`q_proj`,`k_proj`,`v_proj`,`o_proj`,`gate_proj`,`up_proj`,`down_proj`} are used.

For auditing, $H(\cdot)$ is instantiated as SHA-256 and one audit entry is recorded per update step (commit frequency = 1). Each audit record is appended as one JSON object per line (JSONL manifest) and linked via the hash chain. To make digests reproducible, SHA-256 inputs are computed over a canonical serialization of the record fields (deterministic key ordering and UTF-8 encoding). Under fixed seeds and deterministic settings, this yields reproducible record digests and verification outcomes, and the auditor-side verification procedure in Section 4.4 can be executed verbatim to validate chain integrity and parameter-digest consistency.

5.3. *E0: Unified Audit Pipeline*

5.3.1. Purpose

The first experiment validates the framework-level stability, integrity, and efficiency of the proposed AuditableLLM audit pipeline. Specifically, it examines whether the hash-chained audit layer can (1) maintain tamper-evident integrity across continuous updates and (2) operate efficiently with negligible runtime and storage overhead. This experiment represents a foundation-level verification of auditability and efficiency, ensuring that all subsequent fine-tuning and compliance operations are built upon a secure and lightweight audit substrate.

5.3.2. Method

The model adaptation process is instrumented with the audit layer described in Section 4. During 1000 simulated update transactions (including training, fine-tuning, and compliance modifications), the audit layer generates and links cryptographic digests using SHA-256 hashing. Two complementary tests are performed:

- E0-A: Technical Integrity. A canonical audit manifest of 1000 entries is created, and three controlled tampering types are introduced: record deletion, record re-ordering, and digest corruption. Each modified manifest is independently verified using the verification suite, which checks chained-hash consistency and traceability. The verifier reports whether tampering is detected, the index of the first corrupted entry, and the total verification time.

- E0-B: Audit Efficiency. The runtime and storage overhead introduced by the audit process are measured relative to a non-audited baseline. Metrics include per-step latency, total slowdown, raw and compressed log size, and verification time as a function of record count T. Each configuration is averaged over five independent runs on the same hardware described in Section 5.2.

5.3.3. Results

Table 4 reports integrity verification on a 1000-record manifest under three tampering strategies (deletion, reordering, and hash corruption). In all 20 trials per strategy, the verifier detected the first invalid record with 100% accuracy, and the average verification time remained stable at 0.037–0.038 s per run. This behavior follows directly from the hash-chain dependency: any ex post modification must alter at least one record digest and therefore breaks the chain unless the adversary can find a collision or second preimage for SHA-256, which is computationally infeasible under standard assumptions. Consequently, detection performance is governed by cryptographic properties of the chain rather than by the underlying model size or the specific update sequence; model size only affects the one-time cost of computing the per-update parameter digest $\mathbf{h}_{\Delta\theta_t}$, not the verifier's $O(T)$ scan.

Table 4. Integrity validation results on a 1000-record audit manifest.

Tamper Type	Detection Rate	Avg. Verify Time
Record deletion	100%	0.037 s
Record reordering	100%	0.038 s
Hash corruption	100%	0.037 s

Table 5 further quantifies the operational cost of enabling audit logging during a 1000-step pipeline. The end-to-end overhead is 3.4 s in total (3.4 ms/step, 5.7% slowdown), and is dominated by digest computation (97.1% of the added time), indicating that the audit layer introduces a thin, predictable cost that is orthogonal to the GPU-bound forward/backward passes. Storage overhead is modest: raw JSONL logs require 0.61 MB (~611 B/record), while gzip reduces this to 0.15 MB (~153 B/record), implying that long-lived audit retention is feasible even under frequent updates. Finally, offline verification scales linearly with the number of records; the measured 0.04 s per 1000 records implies sub-second verification for typical log sizes in the evaluated setting (e.g., on the order of 10^4 records), enabling practical on-demand audits without adding online training latency when verification is not invoked.

Table 5. Audit efficiency analysis on 1000 training steps.

Metric	Value	Per Step	Main Contributor
Time overhead	+3.4 s	3.4 ms	Hash computation (97.1%)
Log size (raw)	0.61 MB	611 B	Digest fields (53.2%)
Log size (gzip)	0.15 MB	153 B	–
Verification (offline)	$O(T)$	0.04 s/1k	Hash recomputation

As visualized in Figure 4, these measurements translate into a small increase in wall-clock training time (approximately 59.6 s without audit vs. 63.0 s with audit over 1000 steps). Taken together, E0 supports two conclusions aligned with the design objectives: (i) the audit trail is tamper-evident with deterministic detection under the stated cryptographic assumptions, and (ii) the resulting accountability comes with low and predictable runtime/storage overhead suitable for routine integration into PEFT-based adaptation and unlearning workflows.

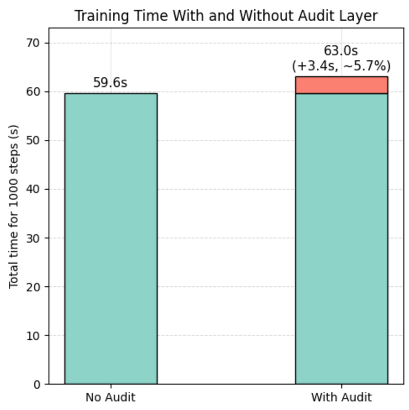

Figure 4. Training time over 1000 steps with and without the audit layer. The main bar indicates baseline training time, and the stacked top segment indicates the additional audit overhead.

5.3.4. Discussion

The results confirm that the AuditableLLM audit layer provides strong structural integrity guarantees with minimal computational or storage overhead. Tamper detection was consistently accurate across all manipulation types, and the reported verification time scales linearly with manifest size (Tables 4 and 5). The per-step runtime increase (3–4 ms) is negligible relative to standard PEFT costs, demonstrating that full auditability can be achieved without sacrificing efficiency. These findings validate the audit layer as a reliable foundation for higher-level experiments on auditable fine-tuning (E1) and compliance-oriented unlearning (E2).

5.4. E1: Auditable Fine-Tuning

5.4.1. Purpose

The second experiment investigates whether the proposed audit layer can be integrated into the fine-tuning process without affecting model performance or convergence. Specifically, this experiment evaluates whether continuous hash-chain logging and digest generation introduce measurable training overhead or interfere with optimization dynamics. This experiment focuses on validating the non-intrusiveness and system-level compatibility of the audit layer within standard LoRA-based fine-tuning workflows.

5.4.2. Method

Following the setup described in Section 5.2, the Llama-3.2-1B-Instruct model is fine-tuned on the MovieLens-small dataset using LoRA adapters under two configurations:

- LoRA Fine-tuning (No Audit)—baseline fine-tuning without audit logging.
- AuditableLLM (Ours)—identical fine-tuning setup with the audit layer enabled, which records operation metadata, cryptographic digests, and hash-chain links at every step.

Both models are trained for 1000 steps with a batch size of 16 and learning rate 2×10^{-4}. Performance is evaluated on the validation split using accuracy and macro-F1, while audit performance is measured by log continuity and runtime overhead. All timing results are averaged over five independent runs.

5.4.3. Results

Table 6 and Figure 5 jointly report an ablation that toggles the audit layer on an otherwise identical LoRA fine-tuning pipeline. Relative to the base model, both LoRA fine-tuning variants yield a large utility gain (Acc: 0.2041 → 0.5539/0.5524; Macro-F1: 0.1928 → 0.4868/0.4840), confirming that the task improvement is driven by the underlying adaptation rather than by auditing. Enabling audit logging changes validation accuracy from 0.5539 to 0.5524 ($\Delta = -0.0015$) and macro-F1 from 0.4868 to 0.4840 ($\Delta = -0.0028$), i.e., differences at the 10^{-3} level in absolute terms. Given the magnitude of the LoRA-induced improvement over the base model, these deltas indicate that audit integration is practically non-intrusive to fine-tuning utility in this setting.

From a systems standpoint, the integrity and cost measurements align with the intended role of AuditableLLM as a lightweight logging overlay: the audit mechanism maintains hash-chain continuity and adds only 3.4 ms per step (5.7% slowdown), which is consistent with CPU-side digesting and metadata appends that do not alter GPU-bound optimization dynamics. Overall, E1 supports the conclusion that AuditableLLM adds verifiable step-level traceability without materially affecting fine-tuning performance, consistent with the low-overhead behavior established in Experiment E0.

Table 6. Comparison of fine-tuning performance with and without the audit layer. Higher scores of Accuracy (Acc) and Macro-F1(F1) indicate better model utility and stability.

Method	Acc	F1
Base Model (No Training)	0.2041	0.1928
LoRA Fine-tuning (No Audit)	0.5539	0.4868
AuditableLLM (Ours)	**0.5524**	**0.4840**

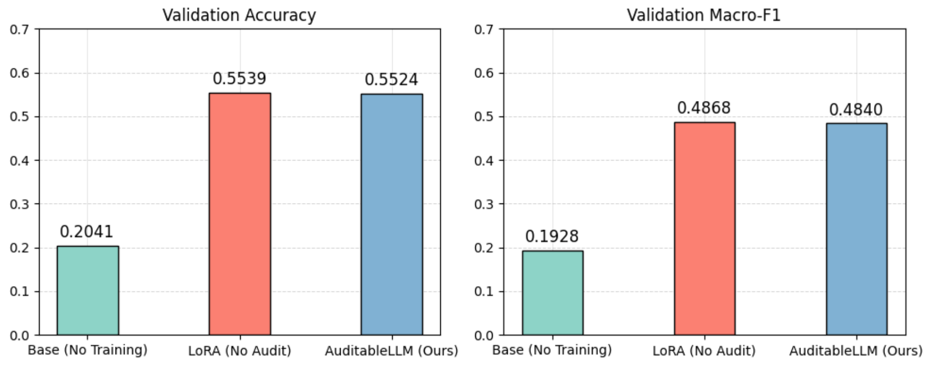

Figure 5. Validation accuracy and macro-F1 on MovieLens-small (100K) for the base model, LoRA without audit, and AuditableLLM with audit logging.

5.4.4. Discussion

The results confirm that AuditableLLM achieves full transparency and reliability during the fine-tuning stage with no loss of model utility or optimization stability. The audit layer consistently maintained tamper-evident linkage across all training steps and introduced only minimal, predictable runtime overhead. Together, these findings demonstrate that the audit layer can be seamlessly embedded into standard fine-tuning pipelines, providing verifiable traceability without compromising performance, forming the foundation for auditable unlearning evaluation in E2.

5.5. E2: Auditable Unlearning

5.5.1. Purpose

The third experiment examines whether the proposed audit layer can be seamlessly integrated into the unlearning process without degrading model performance. Specifically, this experiment evaluates whether the inclusion of audit logging and hash-chain verification affects the forgetting-retention balance achieved by the underlying unlearning algorithm.

This experiment therefore focuses on verifying the non-intrusiveness and compatibility of the audit layer within influence-based parameter-efficient unlearning.

5.5.2. Method

Following the same setup as in Section 5.2, 10% of the training samples are designated as the forget set $\mathcal{D}_{\text{forget}}$, and the remaining 90% as the retain set $\mathcal{D}_{\text{retain}}$. The Llama-3.2-1B-Instruct model is first fine-tuned on the complete dataset.

Unlearning is then performed using an approximate influence-based parameter correction method on the LoRA adapter parameters, following the unified parameter-efficient unlearning algorithm of [34]. Both an unaudited run and an audited run are executed, with the audit layer enabled only in the latter.

A simplified instance-removal instantiation is used, where examples in $\mathcal{D}_{\text{forget}}$ are treated as the deleted set. The corresponding adapter-parameter change is approximated by solving the associated quadratic subproblem over the LoRA weights. This approximation is computed with a small number of mini-batch updates using Hessian-vector products, leveraging the mini-batch subset $\mathcal{D}_{\text{mini}}$ introduced in Section 5.2. The resulting aggregate update $\Delta\theta_{\text{unlearn}}$ is added to the previously fine-tuned adapter to obtain the unlearned model, rather than fully retraining from scratch.

Within the AuditableLLM condition, this unlearning run is represented as a sequence of update events, and each audited unlearning step produces a cryptographic digest, timestamp, and hash-chain link that is verified at the end of the run, together with an operation-type flag ("unlearning") and a reference to the identifier of the associated forget set.

The following four configurations are compared:

- Full Training ($R \cup F$)—baseline model trained on all data.
- Retrain (R only)—model retrained from scratch excluding deleted data.
- Influence-PEFT (No Audit)—standard influence-based unlearning baseline.
- AuditableLLM (Ours)—same unlearning process with audit layer integration.

Performance is evaluated on $\mathcal{D}_{\text{retain}}$ and $\mathcal{D}_{\text{forget}}$ using accuracy and macro-F1 to quantify retention and forgetting respectively.

5.5.3. Results

Table 7 evaluates whether enabling auditing perturbs the underlying influence-based unlearning dynamics. Relative to the non-audited Influence–PEFT baseline, AuditableLLM yields essentially identical utility on $\mathcal{D}_r$: accuracy changes from 0.4813 to 0.4807 ($\Delta = -0.0006$) and macro-F1 from 0.3201 to 0.3192 ($\Delta = -0.0009$), i.e., differences at the 10^{-3} level. Meanwhile, forgetting behavior is preserved: $\mathcal{D}_f$ accuracy remains low ($0.2435 \rightarrow 0.2443$, $\Delta = +0.0008$), and stays close to the retrain-on-R reference (0.2386), indicating that the retain–forget trade-off is not materially altered by the audit layer.

Compared to Full Training on $R \cup F$, both retraining and unlearning markedly reduce $\mathcal{D}_f$ accuracy ($0.5279 \rightarrow 0.2386$–$0.2443$) while retaining reasonable performance on $\mathcal{D}_r$, and AuditableLLM tracks the same behavior as Influence–PEFT. Hash-chain verification maintains full digest consistency for the unlearning log and can be executed in a single linear pass over the records, supporting the interpretation that auditing acts as an orthogonal, low-overhead wrapper around the chosen PEFT unlearning algorithm.

Table 7. Comparison of unlearning performance with and without the audit layer. Higher scores on $\mathcal{D}_r$ indicate better utility, while lower scores on $\mathcal{D}_f$ indicate stronger forgetting.

Method	**Acc ($\mathcal{D}_r$)**	**F1 ($\mathcal{D}_r$)**	**Acc ($\mathcal{D}_f$)**
Full Training ($R \cup F$)	0.5512	0.3844	0.5279
Retrain (R only)	0.4925	0.3270	0.2386
Influence-PEFT (No Audit)	0.4813	0.3201	0.2435
AuditableLLM (Ours)	**0.4807**	**0.3192**	**0.2443**

Figures 6 and 7 provide a complementary view of the same conclusions. Figure 6 contrasts retain and forget accuracy: the audited and non-audited Influence–PEFT configurations are nearly overlapping on both $\mathcal{D}_r$ and $\mathcal{D}_f$, while both remain far below the Full Training forget accuracy, reflecting effective forgetting. Figure 7 shows the retain-set macro-F1, where AuditableLLM again tracks the non-audited baseline closely. Overall, E2 indicates that the audit layer can be integrated with influence-based unlearning without introducing measurable changes to the unlearning outcome, while providing verifiable step-level traceability of the update history.

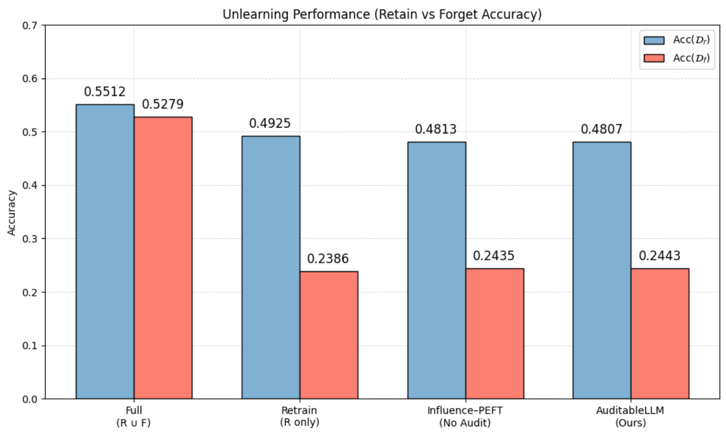

Figure 6. Unlearning performance (accuracy) on MovieLens-small (100 K) for the full model, retrain baseline, influence-based unlearning, and AuditableLLM.

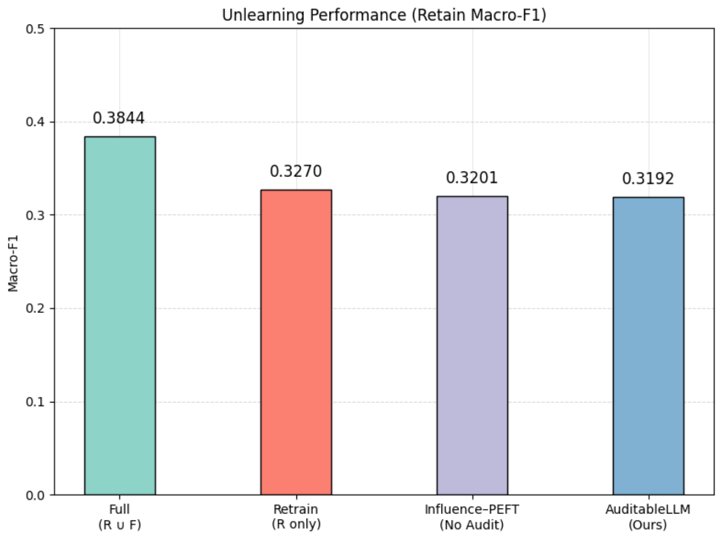

Figure 7. Unlearning performance (macro-averaged F1) on MovieLens-small (100 K) for the same configurations as in Figure 6.

5.5.4. Discussion

The results validate that AuditableLLM achieves algorithmic correctness and regulatory accountability without sacrificing performance. In this experiment, the unlearning dynamics are driven by the unified parameter-efficient unlearning algorithm of [34]; the audit layer is orthogonal to this choice and can be wrapped around alternative unlearning algorithms without modification. The influence-based unlearning process removes targeted data effects effectively, and the integrated audit layer guarantees cryptographic traceability for each update step. The negligible differences between the audited and non-audited variants confirm that accountability and efficiency can coexist within a unified pipeline. Together, these findings demonstrate that AuditableLLM enables transparent, verifiable, and compliance-ready model unlearning with no observable loss in performance. This discussion emphasizes that E2 focuses on behavioral utility and forgetting metrics on $\mathcal{D}_{\text{retain}}$ and $\mathcal{D}_{\text{forget}}$; membership-privacy aspects of unlearning are evaluated separately in Experiment E3. In addition, the empirical study is restricted to a single parameter-efficient unlearning baseline in order to isolate the impact of integrating the audit layer; a broader empirical comparison against multiple state-of-the-art unlearning algorithms is left to future work.

5.6. E3: Membership Inference Robustness

5.6.1. Purpose

The fourth experiment complements the behavioral unlearning evaluation in E2 by examining whether the unlearned models reduce membership leakage on the forget set $\mathcal{D}_{\text{forget}}$. Concretely, this experiment asks whether an adversary performing a loss-based membership inference attack can still distinguish samples in $\mathcal{D}_{\text{forget}}$ from held-out non-members, and how this attack performance compares to a model retrained from scratch on $\mathcal{D}_{\text{retain}}$ only. This experiment focuses on the privacy dimension of unlearning and assesses whether the audit layer affects the membership footprint of the underlying unlearning algorithm.

5.6.2. Method

The four configurations from Experiment E2 are reused:

- Full Training ($R \cup F$)—baseline model trained on all data.
- Retrain (R only)—model retrained from scratch excluding deleted data.
- Influence–PEFT (No Audit)—influence-based unlearning baseline.
- AuditableLLM (Ours)—same unlearning process with audit layer integration.

For each configuration, a membership inference task is constructed in which points from $\mathcal{D}_{\text{forget}}$ (as used in E2) act as members and an equal number of samples from the held-out test split act as non-members. Following prior work on loss-based MIAs [28,29], the adversary observes the per-example training loss $\ell_M(x, y)$ and uses its negative value $s(x, y) = -\ell_M(x, y)$ as a membership score. A threshold τ is calibrated on a separate calibration subset by maximizing attack accuracy, and the resulting classifier is evaluated on an independent evaluation subset. The ROC-AUC of this score, the attack accuracy, and the membership advantage $\text{Adv} = \text{TPR} - \text{FPR}$ on $\mathcal{D}_{\text{forget}}$ are reported, averaged over five independent runs.

5.6.3. Results

Table 8 evaluates membership leakage on the forget set $\mathcal{D}_{\text{forget}}$ using a loss-based MIA. As expected, the fully trained model exhibits a clear membership signal (AUC 0.66, attack accuracy 0.62, advantage 0.24). Retraining on $\mathcal{D}_{\text{retain}}$ substantially weakens this signal, bringing the attack close to chance (AUC 0.54, accuracy 0.53, advantage 0.06), which is consistent with effective removal of membership traces for the forget set.

The influence-based unlearning baseline mitigates leakage relative to full training but does not reach the retrain reference (AUC 0.59, accuracy 0.59, advantage 0.16), reflecting the typical utility–privacy trade-off of approximate unlearning. Crucially, enabling auditing does not materially change these privacy outcomes: AuditableLLM matches the baseline at the reported granularity (AUC 0.59, advantage 0.16) and differs by only 0.01 in attack accuracy ($0.59 \rightarrow 0.58$). Under the stated honest-but-curious threat model and this loss-based evaluation, the audit layer therefore behaves as a privacy-neutral wrapper: it preserves the unlearning algorithm's membership-risk profile while adding a verifiable trace of update operations. Figure 8 provides a visual summary of the same comparisons.

Table 8. Loss-based membership inference attack performance on the forget set $\mathcal{D}_{\text{forget}}$ for MovieLens-small. Lower values indicate weaker membership inference and better privacy (i.e., closer to random guessing).

Method	$\text{AUC}^{F}_{\text{MIA}}$	$\text{Acc}^{F}_{\text{MIA}}$	$\text{Adv}^{F}_{\text{MIA}}$
Full Training ($R \cup F$)	0.66	0.62	0.24
Retrain (R only)	0.54	0.53	0.06
Influence–PEFT (No Audit)	0.59	0.59	0.16
AuditableLLM (Ours)	**0.59**	**0.58**	**0.16**

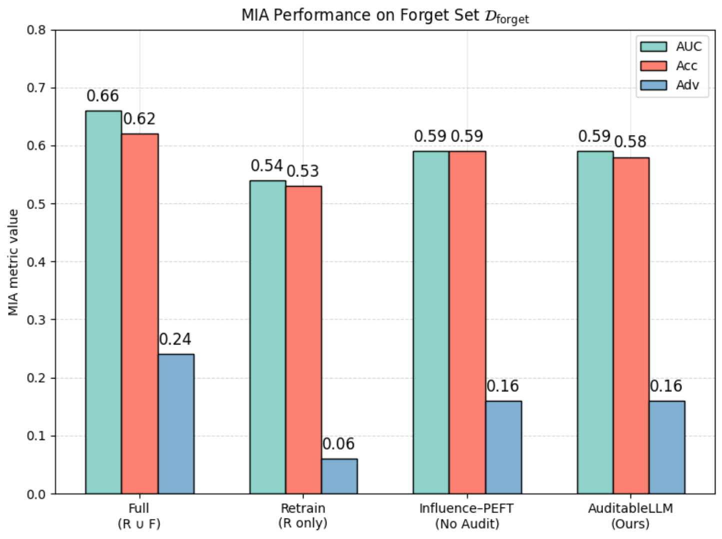

Figure 8. Loss-based membership inference attack performance on the forget set $\mathcal{D}_{\text{forget}}$ for the four configurations in Experiment E3.

5.6.4. Discussion

The MIA results provide a more nuanced view of unlearning effectiveness. On the one hand, both influence-based unlearning configurations substantially reduce membership leakage on $\mathcal{D}_{\text{forget}}$ compared to the fully trained model, confirming that approximate parameter-efficient unlearning can meaningfully weaken membership signals in practice. On the other hand, the unlearned models remain more vulnerable to membership inference than the retrained baseline, which is consistent with the approximate nature of the underlying unlearning algorithm. Crucially, the audit layer has no observable effect on MIA performance: the audited and non-audited variants are indistinguishable within 0.01 across all reported metrics, indicating that cryptographic logging and hash-chain verification can be integrated without introducing additional membership leakage. Overall, E3 shows that, in this experimental setting, AuditableLLM inherits the membership privacy properties of the underlying unlearning algorithm while providing verifiable traceability for unlearning operations.

6. Conclusions and Future Work

This paper introduced AuditableLLM, a unified and lightweight framework for auditable model adaptation in LLMs. Rather than targeting a specific unlearning algorithm, AuditableLLM generalizes the notion of auditability across the entire model lifecycle, covering fine-tuning, continual updates, and compliance-driven deletion. The framework augments existing parameter-efficient adaptation pipelines (e.g., LoRA/QLoRA, adapter-tuning, influence-based updates) with a modular, hash-chained audit layer that aims to ensure transparent, verifiable, and tamper-evident recordkeeping under the stated honest-but-curious provider assumption without modifying the core training algorithm.

In the current instantiation, AuditableLLM is evaluated as a proof-of-concept on a 1B-parameter LLM under an honest-but-curious provider assumption. Within this setting, comprehensive experiments (E0–E3) demonstrate that the proposed audit layer can be seamlessly integrated into fine-tuning and unlearning phases: it preserves model performance with negligible overhead, maintains hash-chain integrity under continuous updates with 100% tamper detection in the conducted experiments, and, under a simple loss-based membership inference attack on the forget set, does not increase membership leakage relative to the non-audited unlearning baseline. These results indicate that efficiency, reliability, accountability, and empirically assessed membership privacy can coexist within a single auditable adaptation framework in benign, small-scale configurations, and motivate further work toward extending these guarantees to adversarial and frontier-scale environments.

Limitations and Future Directions

While AuditableLLM provides a practical foundation for more transparent and compliance-aware model management under the stated threat model, several open challenges remain.

- **Dataset scale and external validity.** The empirical evaluation is conducted in a small-to-mid-scale setting and uses a lightweight benchmark to enable repeated, instrumented adaptation and unlearning runs. Therefore, the current results primarily substantiate the feasibility, traceability, and overhead characteristics of the audit layer. The auditing mechanism itself is designed to attach to update transactions and its cost is driven primarily by the volume of logged update events and hash evaluations rather than the dataset size alone.
- **Scalability and deployment.** Extending the audit abstraction to frontier-scale LLMs and real-time MLaaS settings. In these environments, asynchronous updates, multi-tenant pipelines, and streaming adaptation introduce additional constraints for both performance and verifiability.
- **Stronger cryptographic assurance.** The present hash-chained audit layer is deliberately positioned as a lightweight, tamper-evident logging mechanism for LoRA-based adaptation under an honest-but-curious model provider. Extending these guarantees to stronger adversaries—for example, a fully malicious provider who trains models offline and later reports a fabricated but self-consistent history—calls for additional roots of trust (e.g., trusted time-stamping services, hardware-secured or remotely attested logging) and, potentially, the integration of heavier cryptographic proof systems such as proofs-of-learning and zero-knowledge proofs of training or unlearning. A natural direction for future work is to explore hybrid constructions in which AuditableLLM supplies the lightweight hash-chained logging substrate, while PoL- or ZK-style at-

testations (or trusted hardware) provide end-to-end origin authentication in the fully malicious setting.

- **Cross-paradigm generalization.** Applying the audit layer to additional model adaptation paradigms, such as prompt tuning, retrieval-augmented generation (RAG), and federated model updates, to validate the framework's interoperability across heterogeneous architectures.
- **Governance, privacy, and longitudinal assurance.** Developing standardized evaluation benchmarks for audit persistence, data influence stability, and multi-cycle reliability. The hash-chain-backed audit layer already provides a chronological, tamper-evident view of all adaptation steps together with metadata that summarizes data provenance and training or unlearning configurations. This structure can be leveraged to investigate anomalous behavior, attribute it to specific updates, and roll back to known-good checkpoints, and it is naturally compatible with poisoning-aware and privacy-aware training pipelines. A promising direction for future work is to systematically integrate AuditableLLM with poisoning-robust and differentially private learning or unlearning algorithms. Such efforts could support the creation of auditable compliance standards bridging technical verification with legal accountability.

Overall, AuditableLLM should be viewed as a lightweight, model-agnostic audit layer that demonstrates the feasibility of hash-chain-backed logging, behaviorally validated and supplemented with a simple loss-based membership inference evaluation, for parameter-efficient adaptation under an honest-but-curious provider and small-scale LLM setting. It takes a first step toward more transparent and regulation-aware LLM management, while full adversarial robustness, frontier-scale deployment, and formally certified privacy and poisoning resilience remain important open problems for future research.

Author Contributions: Conceptualization, D.L. and G.Y.; methodology, D.L.; software, D.L.; validation, D.L.; formal analysis, D.L.; investigation, D.L.; resources, G.Y., X.W. and B.L.; data curation, D.L.; writing—original draft preparation, D.L.; writing—review and editing, G.Y., X.W. and B.L.; visualization, D.L.; supervision, G.Y., X.W. and B.L.; project administration, G.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: This study used only publicly available datasets (MovieLens-small). Processed data and source code are not yet publicly released because the GitHub (https://github.com/) repository is under maintenance, but they can be obtained from the corresponding authors on request.

Acknowledgments: The authors gratefully acknowledge the technical support provided by the UTS eResearch and HPC support teams during the computational experiments. During the preparation of this manuscript, the authors used OpenAI ChatGPT (https://chatgpt.com/, accessed on 18 December 2025) to assist in drafting and refining certain sections. All content has been reviewed and edited by the authors, who take full responsibility for the final version of the manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 (General Data Protection Regulation)*; Official Journal of the European Union: Luxembourg, 2016; pp. 1–88.
2. Liu, B.; Ding, M.; Shaham, S.; Rahayu, W.; Farokhi, F.; Lin, Z. When Machine Learning Meets Privacy: A Survey and Outlook. *ACM Comput. Surv.* **2022**, *54*, 1–36. [CrossRef]

3. Rodríguez, E.; Otero, B.; Canal, R. A Survey of Machine and Deep Learning Methods for Privacy Protection in the Internet of Things. *Sensors* **2023**, *23*, 1252. [CrossRef]
4. Feretzakis, G.; Papaspyridis, K.; Gkoulalas-Divanis, A.; Verykios, V.S. Privacy-Preserving Techniques in Generative AI and Large Language Models: A Narrative Review. *Information* **2024**, *15*, 697. [CrossRef]
5. Bi, T.; Yu, G.; Wang, Q. Privacy in Foundation Models: A Conceptual Framework for System Design. *arXiv* **2024**, arXiv:2311.06998.
6. Ma, B.; Jiang, Y.; Wang, X.; Yu, G.; Wang, Q.; Sun, C.; Li, C.; Qi, X.; He, Y.; Ni, W.; et al. SoK: Semantic Privacy in Large Language Models. *arXiv* **2025**, arXiv:2506.23603. [CrossRef]
7. Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; Gelly, S. Parameter-Efficient Transfer Learning for NLP. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019; Volume 97, pp. 2790–2799.
8. Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. LoRA: Low-rank Adaptation of Large Language Models. In Proceedings of the International Conference on Learning Representations, Virtual, 25–29 April 2022.
9. Dettmers, T.; Pagnoni, A.; Holtzman, A.; Zettlemoyer, L. QLoRA: Efficient Finetuning of Quantized Llms. In *Advances in Neural Information Processing Systems*; Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2023; Volume 36, pp. 10088–10115.
10. Blanco-Justicia, A.; Jebreel, N.; Manzanares, B.; Sánchez, D.; Domingo-Ferrer, J.; Collell, G.; Tan, K.E. Digital Forgetting in Large Language Models: A Survey of Unlearning Methods. *Artif. Intell. Rev.* **2025**, *58*, 90. [CrossRef]
11. Sai, S.; Mittal, U.; Chamola, V.; Huang, K.; Spinelli, I.; Scardapane, S.; Tan, Z.; Hussain, A. Machine Un-learning: An Overview of Techniques, Applications, and Future Directions. *Cogn. Comput.* **2024**, *16*, 482–506. [CrossRef]
12. Chen, A.; Li, Y.; Zhao, C.; Huai, M. A Survey of Security and Privacy Issues of Machine Unlearning. *AI Mag.* **2025**, *46*, e12209. [CrossRef]
13. Liu, S.; Yao, Y.; Jia, J.; Casper, S.; Baracaldo, N.; Hase, P.; Yao, Y.; Liu, C.Y.; Xu, X.; Li, H.; et al. Rethinking Machine Unlearning for Large Language Models. *Nat. Mach. Intell.* **2025**, *7*, 181–194. [CrossRef]
14. Jiang, Y.; Yu, G.; Wang, Q.; Wang, X.; Ma, B.; Sun, C.; Ni, W.; Liu, R.P. Split Unlearning. In Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, Taipei, Taiwan, 13–17 October 2025; pp. 948–962. [CrossRef]
15. Jia, H.; Yaghini, M.; Choquette-Choo, C.A.; Dullerud, N.; Thudi, A.; Chandrasekaran, V.; Papernot, N. Proof-of-Learning: Definitions and Practice. In Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 24–27 May 2021; pp. 1039–1056. [CrossRef]
16. Fang, C.; Jia, H.; Thudi, A.; Yaghini, M.; Choquette-Choo, C.A.; Dullerud, N.; Chandrasekaran, V.; Papernot, N. Proof-of-Learning Is Currently More Broken Than You Think. In Proceedings of the 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P), Delft, The Netherlands, 3–7 July 2023; pp. 797–816. [CrossRef]
17. Wang, L.; Chen, S.; Jiang, L.; Pan, S.; Cai, R.; Yang, S.; Yang, F. Parameter-Efficient Fine-Tuning in Large Language Models: A Survey of Methodologies. *Artif. Intell. Rev.* **2025**, *58*, 227. [CrossRef]
18. Nguyen, T.T.; Huynh, T.T.; Ren, Z.; Nguyen, P.L.; Liew, A.W.C.; Yin, H.; Nguyen, Q.V.H. A Survey of Machine Unlearning. *ACM Trans. Intell. Syst. Technol.* **2025**, *16*, 1–46. [CrossRef]
19. Le-Khac, U.N.; Truong, V.N. A Survey on Large Language Models Unlearning: Taxonomy, Evaluations, and Future Directions. *Artif. Intell. Rev.* **2025**, *58*, 399. [CrossRef]
20. Liu, Z.; Jiang, Y.; Shen, J.; Peng, M.; Lam, K.Y.; Yuan, X.; Liu, X. A Survey on Federated Unlearning: Challenges, Methods, and Future Directions. *ACM Comput. Surv.* **2025**, *57*, 1–38. [CrossRef]
21. Garg, S.; Goel, A.; Jha, S.; Mahloujifar, S.; Mahmoody, M.; Policharla, G.V.; Wang, M. Experimenting with Zero-Knowledge Proofs of Training. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, Copenhagen Denmark, 26–30 November 2023; pp. 1880–1894. [CrossRef]
22. Peng, Z.; Xu, J.; Chu, X.; Gao, S.; Yao, Y.; Gu, R.; Tang, Y. VFChain: Enabling Verifiable and Auditable Federated Learning via Blockchain Systems. *IEEE Trans. Netw. Sci. Eng.* **2022**, *9*, 173–186. [CrossRef]
23. Truong, N.; Sun, K.; Wang, S.; Guitton, F.; Guo, Y. Privacy Preservation in Federated Learning: An Insightful Survey from the GDPR Perspective. *Comput. Secur.* **2021**, *110*, 102402. [CrossRef]
24. Graves, L.; Nagisetty, V.; Ganesh, V. Amnesiac Machine Learning. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 11516–11524. [CrossRef]
25. Jiang, Y.; Shen, J.; Liu, Z.; Tan, C.W.; Lam, K.Y. Toward Efficient and Certified Recovery From Poisoning Attacks in Federated Learning. *IEEE Trans. Inf. Forensics Secur.* **2025**, *20*, 2632–2647. [CrossRef]
26. Das, B.C.; Amini, M.H.; Wu, Y. Security and Privacy Challenges of Large Language Models: A Survey. *ACM Comput. Surv.* **2025**, *57*, 1–39. [CrossRef]
27. Wang, W.; Tian, Z.; Liu, A.; Yu, S. TAPE: Tailored Posterior Difference for Auditing of Machine Unlearning. In Proceedings of the ACM on Web Conference 2025, Sydney, NSW, Australia, 28 April–2 May 2025; pp. 3061–3072. [CrossRef]
28. Shokri, R.; Stronati, M.; Song, C.; Shmatikov, V. Membership Inference Attacks Against Machine Learning Models. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 3–18. [CrossRef]

29. Nasr, M.; Shokri, R.; Houmansadr, A. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 739–753. [CrossRef]
30. Koh, P.W.; Liang, P. Understanding Black-Box Predictions via Influence Functions. In Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; Volume 70, pp. 1885–1894.
31. Eisenhofer, T.; Riepel, D.; Chandrasekaran, V.; Ghosh, E.; Ohrimenko, O.; Papernot, N. Verifiable and Provably Secure Machine Unlearning. In Proceedings of the 2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML), Copenhagen, Denmark, 9–11 April 2025; pp. 479–496. [CrossRef]
32. Shaik, T.; Tao, X.; Xie, H.; Li, L.; Zhu, X.; Li, Q. Exploring the Landscape of Machine Unlearning: A Comprehensive Survey and Taxonomy. *IEEE Trans. Neural Netw. Learn. Syst.* **2025**, *36*, 11676–11696. [CrossRef]
33. Thudi, A.; Jia, H.; Shumailov, I.; Papernot, N. On the Necessity of Auditable Algorithmic Definitions for Machine Unlearning. In Proceedings of the 31st USENIX Security Symposium (USENIX Security 22), Boston, MA, USA, 10–12 August 2022; pp. 4007–4022.
34. Ding, C.; Wu, J.; Yuan, Y.; Lu, J.; Zhang, K.; Su, A.; Wang, X.; He, X. Unified Parameter-Efficient Unlearning for LLMs. In Proceedings of the Thirteenth International Conference on Learning Representations, Singapore, 24–28 April 2025.
35. Zhang, Y.; Hu, Z.; Bai, Y.; Wu, J.; Wang, Q.; Feng, F. Recommendation Unlearning via Influence Function. *ACM Trans. Recomm. Syst.* **2025**, *3*, 1–23. [CrossRef]
36. Warnecke, A.; Pirch, L.; Wressnegger, C.; Rieck, K. Machine Unlearning of Features and Labels. In Proceedings of the 2023 Network and Distributed System Security Symposium, San Diego, CA, USA, 27 February–3 March 2023. [CrossRef]
37. Wichert, L.; Sikdar, S. Rethinking Evaluation Methods for Machine Unlearning. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, FL, USA, 12–16 November 2024; pp. 4727–4739. [CrossRef]
38. Harper, F.M.; Konstan, J.A. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst. (TiiS)* **2015**, *5*, 19:1–19:19. [CrossRef]
39. Meta AI. Llama-3.2-1B-Instruct. Hugging Face Model Card, 2024. Available online: https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct (accessed on 15 December 2025).