VeriDP: Verifiable Differentially Private Training

Behzad Abdolmaleki
University of Sheffield
Sheffield, United Kingdom
behzad.abdolmaleki@sheffield.ac.uk

Amir R. Asadi
University of Cambridge
Cambridge, United Kingdom
asadi@statslab.cam.ac.uk

Vahid R. Asadi
University of Waterloo
Waterloo, Canada
vrasadi@uwaterloo.ca

Stefan Köpsell
Barkhausen Institut
Dresden, Germany
stefan.koepsell@barkhauseninstitut.org

Bhavish Mohee
University of Sheffield
Sheffield, United Kingdom
bmohee1@sheffield.ac.uk

Nahid Roustaeifar
University of Sheffield
Sheffield, United Kingdom
nroustaeifar1@sheffield.ac.uk

Maryam Zarezadeh
Barkhausen Institut
Dresden, Germany
maryam.zarezadeh@barkhauseninstitut.org

Abstract

Stochastic Gradient Descent (SGD) is the foundation of modern machine learning (ML). In privacy-sensitive settings, gradients can reveal details about individual data points. Differential Privacy (DP) protects sensitive data during ML training by clipping gradients and adding calibrated Gaussian noise. However, existing frameworks assume semi-honest participants, which fails in adversarial or federated environments where malicious actors can bypass or alter the noise addition process, breaking privacy guarantees.

We present VeriDP, a framework for verifiable differentially private training that cryptographically enforces and proves the correct execution of differentially private stochastic gradient descent (DP-SGD) in zero knowledge. VeriDP integrates Zero-Knowledge Proofs (ZKPs) with polynomial commitments, sumcheck and GKR-based proofs, and incrementally verifiable computation (IVC) to generate compact proofs of correct gradient computation, clipping, averaging, and Gaussian noise generation—without revealing private data or randomness. Unlike previous systems that only verify the final privacy budget, VeriDP enables per-iteration verifiability of each model update, providing strong privacy assurances even in adversarial settings. This establishes a novel and complete Zero-Knowledge Proof of Differentially Private Stochastic Gradient Descent (ZK-DPSGD), uniting differential privacy and verifiable computation for secure and auditable ML. Our evaluation shows that prover time increases linearly with the number of input samples, while both verifier time (2–5 ms) and proof size (3–4 KB) remain compact and effectively constant.

Keywords

Zero-Knowledge Proof, Differential Privacy, Verifiable Machine Learning, Stochastic Gradient Descent

Proceedings on Privacy Enhancing Technologies 2026(3), 48–64

https://doi.org/10.56553/popets-2026-0070

1 Introduction

Machine learning (ML) models are commonly trained using Stochastic Gradient Descent (SGD), a foundational algorithm that underpins many state-of-the-art applications. However, in privacy-sensitive settings, gradients may reveal information about individual training examples, enabling attacks such as membership inference and gradient inversion that can reconstruct or identify individual data points from shared updates [32, 44, 53]. Differential Privacy (DP) has emerged as a principled solution to this challenge: by clipping gradients and adding calibrated noise, mechanisms such as DP-SGD [2] provide formal privacy guarantees that limit the leakage of individual data points. Despite these guarantees, current DP training frameworks typically assume a semi-honest adversarial model in which all participants follow the protocol faithfully. In distributed or federated environments, this assumption may fail. A malicious participant could, for example, manipulate or omit the noise addition step, thereby violating the claimed privacy guarantees without detection. This gap highlights the need for a mechanism that can not only enforce differential privacy, but also provide cryptographically verifiable evidence that the protocol was executed correctly. To tackle this, we propose VeriDP, a framework that combines differential privacy with Zero-Knowledge Proofs (ZKPs). This combination allows for verifiable, privacy-preserving training in challenging environments. Our method guarantees that each DP-SGD iteration, which includes gradient clipping, noise generation, and weight updates, is done correctly and can be proven, all without exposing any private data or randomness.

In addition to the framework itself, we introduce a new formal definition: the Zero-Knowledge Proof of Differentially Private Stochastic Gradient Descent (ZK-DPSGD). This definition clearly outlines the requirements for proving the correct and private execution of DP-SGD under zero-knowledge conditions. We present this definition in Section 4.1. It lays the groundwork for verifiable differential privacy and shapes the design and security assessment of VeriDP.

Problem Statement. We consider settings (including distributed or federated training) in which a prover executes DP-SGD on a committed dataset, but may behave maliciously by deviating from

the prescribed clipping or noise generation steps. The verifier's goal is not only to check a final privacy budget, but to obtain cryptographic assurance that every individual training iteration satisfies the DP-SGD update rule. Formally, the challenge is to design a zero-knowledge protocol that proves, for each iteration t, that (i) per-example gradients were computed on elements of the committed dataset, (ii) gradient norms were correctly clipped to a public bound C, and (iii) Gaussian noise sampled from the prescribed distribution was correctly generated and added–without revealing the data, gradients, or randomness. Existing verifiable DP systems provide only aggregate certification of a completed training run, leaving intermediate updates vulnerable to undetectable adversarial deviation. Our work addresses this gap by enabling per-iteration recursive verification of DP-SGD execution under active adversaries.

Our Contributions. In this work, we introduce VeriDP, the first framework to provide publicly verifiable, per-iteration enforcement of DP-SGD under zero knowledge. While prior systems such as Confidential-DPproof [43] certify only a single, monolithic training run, VeriDP enables continuous and composable verification of every iteration. Our main contributions are:

- **IVC-based Verifiable DP-SGD.** We design the first incrementally verifiable DP-SGD construction. Each SGD iteration produces a proof that is recursively merged into a constant-size certificate. Proof size and verifier time remain constant regardless of training length—unlike monolithic-circuit approaches (e.g., Confidential-DPproof [43]), whose proof size and verification cost grow linearly with the number of iterations.
- **Per-Iteration Privacy Enforcement.** VeriDP proves, for every iteration, that gradient clipping, Gaussian noise addition, and model updates follow the DP-SGD rules. This enables per-iteration attestation of privacy, rather than an aggregate certificate for the entire training run.
- **Composable Recursive Proof Architecture.** Each iteration's proof forms the input state for the next proof, ensuring that the entire training trajectory is verifiable, consistent, and tamper-resistant. This recursive structure enables decentralized, federated, and multi-party use cases where updates must be individually accountable.
- **Publicly Verifiable Commitments Integrated with Recursion.** VeriDP uses polynomial commitments that are publicly verifiable and compatible with recursion. Unlike the Information-Theoretic Message Authentication Code (IT-MAC) commitments in Confidential-DPproof [43]–which provide private auditor-only integrity and are not hiding–ours support public verification, long-term auditability, and seamless integration with IVC.
- **End-to-End Verifiable DP-SGD with Dataset Binding and Modular Architecture.** VeriDP provides an end-to-end verifiable DP-SGD pipeline that cryptographically proves every privacy-critical operation—including per-example gradient computation, clipping, averaging, Gaussian noise generation via a GKR-based Box–Muller circuit, noise addition, and weight updates—while ensuring that all DP-SGD operations operate exclusively on the declared dataset through Merkle-based membership proofs. The system integrates tight privacy accounting following the approach of [33], achieves constant-size proofs and constant-time verification through Incrementally Verifiable Computation (IVC) (3–4 KB proofs and sub-5 ms verification), and offers detailed prover-time analysis demonstrating linear scaling with batch size and model complexity. Its modular design cleanly separates the ML, proving, and recursive aggregation layers, enabling straightforward extension to new models and training frameworks.

Together, these contributions establish a new class of verifiable ML protocols that ensure continuous, per-iteration differential privacy with cryptographic soundness and public auditability.

Auditability in Federated Learning. Federated learning (FL) [22, 31] distributes training across clients who share updates without exposing raw data. While this improves data locality, it makes enforcement of privacy mechanisms difficult: a verifier sees only model updates and cannot directly check whether each client correctly applied gradient clipping, subsampling, and calibrated noise as required by DP-SGD [2].

Why per-iteration auditability matters in FL. Training in FL proceeds over many rounds with varying participants and environments [22]. Here, we distinguish between a DP-SGD iteration, which denotes a single gradient update as defined in Definition 2 and Protocol VeriDP (Fig. 1), and a federated round, which refers to a communication cycle that may consist of multiple local DP-SGD iterations executed by a client. All cryptographic verification guarantees in VeriDP are enforced at the level of individual DP-SGD iterations. Per-iteration auditability provides three key benefits: (1) *round-level accountability*, enabling detection and localization of deviations (e.g., skipped clipping or reduced noise); (2) stronger privacy assurance, since every update must satisfy DP-SGD [2] and cannot leak information early while passing only final checks; and (3) support for *public and long-term auditing* in cross-silo or regulated deployments, as increasingly required in high-stakes or regulated ML systems [8].

Why prior guarantees are insufficient. First, *monolithic verification* certifies only the final outcome or overall privacy budget and cannot localize which rounds or clients violated the protocol [43]; selective early-round weakening can remain hidden within aggregate accounting. Second, naive per-iteration verification scales poorly, with proof and verification costs growing linearly in the number of training iterations (and in the number of participating clients in federated settings). In such approaches, each iteration produces an independent proof that must be verified separately, causing total verification time and proof size to grow linearly with the number of iterations. In contrast, VeriDP uses IVC to recursively fold per-round proofs into a single constant-size certificate. Although correctness is enforced at every DP-SGD iteration, the verifier checks only one final proof, so verification time and proof size remain constant with respect to the number of iterations. Publicly verifiable DP noise further requires binding randomness to execution; otherwise, clients may bias noise by grinding seeds while still producing plausible updates.

How VeriDP enables per-iteration auditability. VeriDP generates a proof per iteration enforcing (i) batch membership in the committed dataset, (ii) correct clipping and averaging, and (iii) correct Gaussian noise generation bound to an unbiased, context-dependent seed. These proofs are incrementally linked, yielding a constant-size certificate for the entire training trajectory while

preserving round-level auditability. In a FL setting, each participating client acts as the prover by generating a ZKP that its local DP-SGD update was computed correctly. The central aggregator (or an external auditor) acts as the verifier and checks each proof before incorporating the corresponding client update into the global aggregation phase. Verification therefore occurs prior to model aggregation, ensuring that only privacy-compliant updates contribute to the global model. We provide a detailed and formal description of this integration in Section 4.6.

Overview of Our Results. We introduce VeriDP, a method for verifiable DP training that ensures correctness even in adversarial environments. Standard differentially private stochastic gradient descent (DP-SGD) protects sensitive data by clipping gradients and adding Gaussian noise, but existing implementations assume semi-honest adversaries who follow the protocol faithfully. In federated or distributed training, malicious participants may bypass or manipulate the noise addition step, thereby invalidating the claimed privacy guarantees. VeriDP addresses this gap by combining DP-SGD with ZKPs to provide cryptographic evidence that each training iteration is executed correctly, without revealing underlying data, gradients, noise values, or model parameters. The protocol operates in three phases: (i) dataset commitment via cryptographic primitives, (ii) unbiased randomness seed generation to prevent selective manipulation of noise, and (iii) ZKP of DP-SGD updates, including gradient computation, clipping at the norm bound, averaging, verifiable Gaussian noise generation via a Box-Muller transform [7] circuit, and final weight updates. These operations are encoded as arithmetic circuits and verified using polynomial commitments, sumcheck protocols, and GKR-based arguments, with scalability ensured through IVC. The resulting framework guarantees completeness, knowledge soundness, zero-knowledge privacy, and strict enforcement of DP constraints. Unlike prior works such as Confidential-DPproof, which certify only the final (ϵ, δ) privacy budget, VeriDP achieves fine-grained, per-iteration verifiability of every update, providing stronger assurances of correctness in adversarial training settings.

2 Related Work

The study of verifiable differential privacy (DP) seeks to bridge the gap between theoretical guarantees and their enforcement in adversarial settings. Classical mechanisms such as DP-SGD [13] assume honest participants, but in distributed or federated scenarios malicious parties may manipulate or omit noise addition. Narayan et al. [35] proposed early cryptographic verification of noisy queries, Biswas and Cormode [5] introduced auditable noise generation with public and private randomness, and Sabater et al. [42] extended verifiability to federated averaging. However, these works focus on queries or aggregate updates rather than fine-grained verification of iterative DP-SGD training. A recent line of work applies ZKPs to certify ML properties. Shamsabadi et al. [43] introduced Confidential-DPproof, encoding an entire DP-SGD run into a single verifiable circuit to certify compliance with a declared (ε, δ) budget. While this ensures global compliance, it produces only an aggregate certificate and treats training atomically, preventing detection of intermediate deviations. In federated settings, a participant could bias early rounds and compensate later while still passing final verification. In contrast, our approach enforces per-iteration verification of gradient computation, clipping, and Gaussian noise addition. This enables checkpoint validation and round-level accountability, and—when combined with incrementally verifiable computation—achieves constant-size proofs and constant-time verification, avoiding the linear overhead of monolithic designs. Recent work further expands verifiable DP. Wei et al. [49] proposed ZKP-based verification for data release, but not iterative training. Keinan et al. [23] showed limits of one-run auditing, and Lokna et al. [27] introduced Delta-Siege for empirical auditing. These methods improve auditing but lack cryptographic soundness. Ma et al. [30] proposed VPFL for federated aggregation [20, 41], yet without enforcing DP noise generation. Formal methods strengthen DP foundations: CertiPriv [4], Fuzzi [51], SampCert [11], CheckDP [48], and Duet [38] verify DP properties at the program level. However, they do not prevent adversarial manipulation at runtime. Empirical techniques such as membership inference attacks [9, 40, 44] and privacy auditing frameworks [19, 28, 36, 45] can detect leakage but provide only lower bounds on privacy loss. Overall, prior work spans query verification, federated aggregation, ZKPs, auditing, and formal verification. Yet adversarial robustness for iterative training remains unresolved. Our framework addresses this gap by combining DP-SGD with per-iteration zero-knowledge verification, ensuring that each training iteration is both privacy-preserving and cryptographically verifiable.

3 Preliminaries

Notation. Let $\mathbb{F}$ be a finite field and λ the security parameter. Define $[n] = \{0, \ldots, n-1\}$. Let negl denote a negligible function. Vectors are written in bold lowercase (e.g., $\mathbf{x}, \mathbf{y}$).

Definition 1. *((ϵ, δ)-Differential Privacy [13]). Let ϵ and δ be non-negative scalars such that $\epsilon, \delta \leq 1$.*

Neighboring datasets. *Two datasets D and D' are said to be neighboring if they differ in the data of at most one individual, i.e., by addition, removal, or replacement of a single training example.*

A mechanism A is said to satisfy (ϵ, δ)-differential privacy if, for any pair of neighboring datasets D and D', and any subset $S \subseteq$ Range(A), the following inequality holds:

$$\Pr[A(D) \in S] \leq \exp(\epsilon) \Pr[A(D') \in S] + \delta.$$

Here, ϵ controls the level of privacy, with smaller values implying stronger privacy protection by limiting how much the mechanism's output can change in response to modifications of a single training example. The parameter δ governs the probability of failure of the privacy guarantee, with smaller values providing stronger protection.

Throughout this work, we adopt differential privacy under the above notion of neighboring datasets.

Privacy Accounting via Rényi Differential Privacy. In this work, we perform privacy accounting using Rényi Differential Privacy (RDP) [2, 33]. Each iteration of DP-SGD satisfies $(\alpha, \varepsilon_\alpha)$-RDP for all orders $\alpha > 1$, where ε_α depends on the noise multiplier σ, the clipping norm C, and the sampling rate. By the composability of RDP [33], the cumulative privacy loss over T iterations is given by

$$\varepsilon_\alpha^{\text{tot}} = \sum_{t=1}^{T} \varepsilon_\alpha^{(t)}.$$

The composed RDP guarantee is converted to an (ε, δ)-differential privacy guarantee using the standard transformation [33]:

$$\varepsilon(\delta) = \min_{\alpha>1} \left(\varepsilon_\alpha^{\text{tot}} + \frac{\log(1/\delta)}{\alpha - 1} \right).$$

Throughout the paper, the reported (ε, δ) values correspond to the tightest bound obtained by optimizing over α.

Zero-Knowledge Arguments [16]. Let R be an NP relation with its corresponding language L_R. A zero-knowledge argument of knowledge for R is defined by a triple of algorithm, (G, P, V). We denote G as the setup algorithm, P as the prover, and V as the verifier. This allows P who know a witness, to convince V of validity of some statement without revealing the witness. See more details in Appendix A.

Polynomial Commitments (PCS). A PCS scheme allows a prover to commit to a polynomial and later prove correct evaluation at a point. We use a standard PCS satisfying hiding, evaluation binding, knowledge soundness, and zero-knowledge properties. For the formal definition, see Appendix A.

The Sumcheck Protocol. The sumcheck protocol, originally introduced by Lund et al. [29], is a fundamental tool in algebraic complexity theory that enables the efficient verification of polynomial summations over structured domains. Consider an ℓ-variate polynomial $f : \mathbb{F}^\ell \to \mathbb{F}$ with a variable-degree bound d. The sumcheck protocol allows a verifier V to delegate the computation of the sum of f over the binary hypercube, namely: $H = \sum_{b\in\{0,1\}^\ell} f(b)$, to a prover P. The protocol proceeds iteratively by reducing the evaluation of f over the entire hypercube to evaluations at a single randomly chosen point $r \in \mathbb{F}^\ell$. The key efficiency aspects of the sumcheck protocol are as follows: The prover's complexity is $O(d^\ell)$, as the prover must evaluate and respond iteratively across ℓ rounds. The verifier's complexity is $O(d\ell)$, making the protocol highly efficient for large polynomials. The proof size is also $O(d\ell)$, ensuring concise proof generation. Additionally, the soundness error is $O(d\ell/|\mathbb{F}|)$, which guarantees that a dishonest prover is unlikely to deceive the verifier. The formal description of the sumcheck protocol is provided in Fig. 6 in Appendix A.

The GKR Protocol. The interactive proof system for layered arithmetic circuits, known as the GKR protocol, was introduced by Goldwasser, Kalai, and Rothblum [17]. The GKR protocol extends the sumcheck protocol to enable succinct verification of layered arithmetic circuits. Consider a layered arithmetic circuit C over a finite field $\mathbb{F}$ with depth d, where layer 0 is the output layer and layer d corresponds to the input layer. Each gate in layer i takes two inputs from layer $i + 1$. Let S_i denote the number of gates in layer i and define $s_i := \lceil \log S_i \rceil$. The wiring predicates add_i and $mult_i$ encode addition and multiplication gates. The protocol verifies circuit evaluation by iteratively applying the sumcheck protocol across layers, reducing verification of $\widetilde{v}_i$ to evaluations of $\widetilde{v}_{i+1}$. A complete algebraic description is provided in Appendix A. To obtain zero-knowledge, zero-knowledge sumchecks [6, 50] and zero-knowledge polynomial commitments can be employed. Applying the Fiat–Shamir transform [15] yields a non-interactive argument. A formal description of the GKR protocol is given in Fig. 7 in Appendix A.

Incrementally Verifiable Computation (IVC). IVC schemes [34, 39, 47] facilitate the succinct verification of computations that evolve iteratively. Let F be a function, and consider a sequence of auxiliary witness inputs $\omega_0, \ldots, \omega_{i-1}$ along with an initial input z_0. The iterative computation is defined as $\forall k \in [i], z_{k+1} = F(z_k, \omega_k)$. An IVC scheme enables a prover to generate a proof π_i incrementally, certifying that there exist witness values $\omega_0, \ldots, \omega_{i-1}$ such that the output z_i was derived correctly from the initial input z_0 according to the function F. We defer the full formal definition to Appendix A.

4 Construction:VeriDP Protocol

4.1 Definition of Zero-Knowledge Proof of Differentially Private Stochastic Gradient Descent

Having defined the underlying primitives of Differential Privacy and Zero-Knowledge Arguments, we now formalize their integration within the training process. This section introduces a new definition–the Zero-Knowledge Proof of Differentially Private Stochastic Gradient Descent (ZK-DPSGD)–which captures the precise conditions under which the execution of DP-SGD can be proven correct and privacy-preserving under zero knowledge. The following definition serves as the foundation for our verifiable training framework, VeriDP.

Definition 2. ***Zero-Knowledge Proof of Differentially Private Stochastic Gradient Descent (ZK-DPSGD).***

Let $\mathcal{A}_{DP\text{-}SGD}$ be a stochastic gradient descent algorithm that satisfies (ϵ, δ)-differential privacy according to Definition 1. Let $R_{DP\text{-}SGD}$ be the NP relation defined as:

$$R_{DP\text{-}SGD} = \Big\{ (x, w) \,\Big|\, x = (D, pp, Com(\theta_0,), \theta_T) \\ w = \big(\{g_t\}_{t=1}^T, \{n_t\}_{t=1}^T\big), \\ \theta_T = \mathcal{A}_{DP\text{-}SGD}(D; pp, \theta_0) \Big\},$$

where:

- *D is the committed training dataset. pp are the public hyperparameters (clipping norm C, learning rate η, noise multiplier σ, batch size b, number of iterations T, loss function L, and fixed-point precision parameter f), The fixed-point precision parameter f is treated as part of the public input and is enforced by the verification circuit. The prover must demonstrate that all noise samples and intermediate arithmetic operations are consistent with this declared precision, making numerical behavior part of the cryptographic statement being proven.*
- *The iteration index t denotes the current DP-SGD iteration and is treated as part of the public input and recursive state. For each iteration, the randomness seed is derived as:*

$$s_t = H(\text{training ID} \,\|\, t \,\|\, \text{batch index } b_{ID} \,\|\, \rho_{\mathcal{D}})$$

  The hash computation is encoded inside the arithmetic circuit and verified as part of the zero-knowledge proof, ensuring that the seed is uniquely bound to the iteration context.
- *θ_0 and θ_T are the initial and final model parameters,*
- *$\{g_t\}_{t=1}^T$ are the clipped per-iteration gradients, and*
- *$\{n_t\}_{t=1}^T$ are Gaussian noise vectors generated from a committed randomness seed.*

A protocol $\Pi = (G, P, V)$ *is a* Zero-Knowledge Proof of DP-SGD *for* $R_{DP\text{-}SGD}$ *if the following properties hold:*

- ***Completeness:*** *For every* $(x, w) \in R_{DP\text{-}SGD}$ *and public parameters* $pp \leftarrow G(1^\lambda)$*, if the prover* P *executes* $\mathcal{A}_{DP\text{-}SGD}$ *honestly, then the verifier* V *always accepts:*

$$\Pr\left[\langle P(pp, w), V(pp)\rangle(x) = 1\right] = 1.$$

- ***Knowledge Soundness:*** *For any PPT adversary* A*, there exists a PPT extractor* ξ_A *such that for all* x *and* $pp \leftarrow G(1^\lambda)$*:*

$$\Pr\left[w \leftarrow \xi_A(pp, x) : (x, w) \in R_{DP\text{-}SGD}\right] \geq \Pr\left[\langle A(pp), V(pp)\rangle(x) = 1\right] - negl(\lambda).$$

  This ensures that a successful prover must possess valid gradients $\{g_t\}$ *and noise vectors* $\{n_t\}$ *consistent with the DP-SGD update rule.*
- ***Zero-Knowledge:*** *There exists a PPT simulator* S_A *such that for every PPT adversary* A*, for all* $(x, w) \in R_{DP\text{-}SGD}$*, and* $pp \leftarrow G(1^\lambda)$*:*

$$View(\langle P(pp, w), A(pp, z)\rangle(x)) \approx S_A(x, z),$$

  meaning that the verifier learns nothing beyond the fact that DP-SGD was executed correctly.
- ***Privacy Enforcement:*** *In every SGD iteration* $t \in [T]$*, the prover must satisfy:*

$$\tilde{g}_t = \frac{1}{b}\sum_{i \in B_t} \hat{g}_i + n_t, \quad \hat{g}_i = \frac{g_i}{\max\left(1, \frac{\|g_i\|_2}{C}\right)}, \quad n_t \sim \mathcal{N}\left(0, \sigma^2 C^2 I\right).$$

  This enforces that each update adheres to the clipping and Gaussian noise mechanism required for (ϵ, δ)*-DP.*

Thus, Π *provides a cryptographic proof that DP-SGD was executed faithfully while revealing no information about the underlying data, gradients, noise, or model parameters.*

Initialization Binding and Base Case Enforcement. The initial model parameters θ_0 are treated as part of the public input to the ZK-DPSGD relation and are cryptographically bound in the base case of the IVC construction. Concretely, the prover publishes a commitment $\mathrm{Com}(\theta_0)$ as part of the initial IVC state z_0. The base proof π_0 verifies knowledge of θ_0 such that $\mathrm{Com}(\theta_0)$ opens correctly and that θ_0 satisfies any declared structural or domain constraints (e.g., dimensionality and parameter bounds).

For every subsequent iteration $t > 0$, the recursive verification enforces that θ_t is derived solely from θ_{t-1} via a correct DP-SGD update as specified in Definition 2 and Protocol VeriDP (cf. Fig. 1). Since the IVC state carries forward $\mathrm{Com}(\theta_0)$ and the chain of proofs $\pi_0, \ldots, \pi_{t-1}$, any deviation from the declared initialization or any attempt to introduce an alternative starting model invalidates the recursive proof. This ensures end-to-end integrity of the entire training trajectory, anchored at the committed initial model parameters.

4.2 System and Threat Model

System Model. VeriDP involves three types of entities: a) Prover (Trainer): The prover executes DP-SGD training on a private dataset and produces ZKPs attesting to the correct execution of each training iteration, including gradient computation, clipping, noise generation, and model updates. b) Verifier (Auditor): The verifier checks the ZKPs and validates that the declared DP-SGD parameters and privacy guarantees were enforced. The verifier does not have access to the training data, gradients, noise values, or internal randomness. c) Public Observers: Because VeriDP proofs are publicly verifiable, any third party can independently verify that the DP-SGD execution satisfies the declared privacy guarantees without additional trust assumptions. This supports decentralized and regulatory auditing.

Threat Model. We consider a malicious prover that may arbitrarily deviate from the prescribed DP-SGD protocol in order to improve model utility or leak private information, while attempting to convince verifiers that the training was privacy-preserving. In particular, the adversary may a) Omit or weaken gradient clipping or noise addition during selected iterations, b) Bias or manipulate randomness used for subsampling or Gaussian noise generation, c) Deviate from declared hyperparameters or selectively violate DP constraints, d) Attempt to produce an accepting proof without faithfully executing DP-SGD, or e) Attempt to alter the underlying dataset across iterations. The prover has full control over the training process and execution environment, but is computationally bounded (PPT). The verifier follows the verification protocol honestly and does not collude with the prover. Security relies on i) collision resistance of the hash function, ii) binding of the dataset commitment $\rho_{\mathcal{D}}$, iii) soundness and knowledge extraction of the underlying zero-knowledge argument system, and iv) soundness of the IVC construction.

Adversarial Goals and Security Guarantees. The adversary's goal is to produce a convincing proof while violating differential privacy guarantees. VeriDP prevents such attacks via knowledge soundness: any prover that produces an accepting proof must possess valid witnesses corresponding to correctly clipped gradients, valid Merkle membership proofs for dataset elements, and properly generated Gaussian noise derived from the context-bound randomness seed. The context-bound seed derivation prevents grinding attacks by ensuring that each iteration admits a unique admissible randomness value determined by public inputs and the committed dataset. Zero-knowledge guarantees that no additional information about the dataset, gradients, or randomness is leaked to the verifier or to public observers.

The above system and threat model formalize the adversarial assumptions under which Definition 2 and the VeriDP protocol (Fig. 1) provide verifiable enforcement of DP-SGD.

4.3 VeriDP Overview

We present VeriDP in Fig. 1, a protocol that securely and efficiently certifies an upper bound on the privacy parameter ε resulting from a DP-SGD training run (see Algorithm 1), removing the need for empirical audits of the model's privacy guarantees. The framework operates as follows:

- The prover publicly announces the target privacy guarantees (ε, δ) before training begins.
- The prover and an auditor agree on the DP-SGD training algorithm (cf. Algorithm 1) and fix all associated hyperparameters

including minibatch size, noise multiplier, clipping norm, learning rate, number of iterations, and the loss function. These are chosen to meet the declared privacy parameters (ε, δ) while maintaining high model utility.
- The prover and auditor then execute our interactive zero-knowledge protocol verifying the claimed privacy guarantee. The entire procedure is captured within a single public circuit, enabling the prover to generate one unified proof showing that all steps of the DP-SGD algorithm were executed correctly (cf. Algorithm 1).

Algorithm 1: Differentially Private Stochastic Gradient Descent (DP-SGD)

Input: Weights $\mathbf{W}_{i-1}$, mini-batch $\mathbf{B}_{i-1}$, clipping norm C, learning rate η, noise multiplier σ, loss function $\mathcal{L}(\cdot)$
Output: Updated weights $\mathbf{W}_i$ for mini-batch $\mathbf{B}_i$

Let $\mathbf{B}_{i-1} = \{(x_j, y_j)\}_{j=1}^{l}$ be the mini-batch of size l.
Compute per-example gradients:
for $j = 1$ ***to*** l **do**
  Compute gradient for sample j: $\mathbf{g}_j = \nabla_{\mathbf{W}_{i-1}} \mathcal{L}(\mathbf{W}_{i-1}, (x_j, y_j))$.
  Gradient clipping: $\hat{\mathbf{g}}_j = \mathbf{g}_j \cdot \min(1, \frac{C}{\|\mathbf{g}_j\|_2})$
Average clipped gradients: $\hat{\tilde{\mathbf{g}}} = \frac{1}{l} \sum_{j=1}^{l} \hat{\mathbf{g}}_j$
Noise addition:
- Generate Gaussian noise $\mathbf{n} \sim \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})$
- Add noise to average gradient: $\tilde{\mathbf{g}} = \hat{\tilde{\mathbf{g}}} + \mathbf{n}$

Update weights: $\mathbf{W}_i = \mathbf{W}_{i-1} - \eta \tilde{\mathbf{g}}$

VeriDP scheme is composed of three main phases: a) Data commitment b) Interactive generation of a private randomness seed c) ZKP of DP-SGD training. Each phase of VeriDP is described in detail below.

4.3.1 Data commitment. Given the training dataset $\mathcal{D}$ and public parameters pp as input, this algorithm outputs a commitment $\rho_{\mathcal{D}}$ to the dataset. The resulting dataset commitment $\rho_{\mathcal{D}}$ becomes a public input to every iteration of the protocol and is checked against each mini-batch via Merkle membership proofs, as enforced in Step 2 of Fig. 1

Dataset Consistency Across Iterations. Let $\rho_{\mathcal{D}}$ denote the Merkle root of the committed dataset. We include $\rho_{\mathcal{D}}$ as part of the public input and as part of the recursive state in the IVC construction. At each iteration t, the prover must supply i) Merkle membership proofs for all samples in the selected batch, and ii) a proof that all per-example gradients were computed only over samples whose leaf hashes correspond to $\rho_{\mathcal{D}}$. The recursive IVC state enforces that $\rho_{\mathcal{D}}$ remains unchanged across iterations. Any modification would invalidate the binding property of the commitment and cause verification failure. Thus, all training iterations are cryptographically bound to a single, immutable dataset commitment.

4.3.2 Randomness seed generation. DP-SGD achieves its privacy guarantees by introducing randomness at multiple points during training. This randomness is calibrated according to how sensitive the model is to individual training examples. Two primary randomization techniques are used: subsampling the training data and adding noise to the clipped gradients. However, a dishonest prover could manipulate these randomness sources to improve model accuracy at the cost of privacy-for instance, by repeatedly running the training with different random seeds and selecting the one that gives the best performance, or by choosing noise values with low magnitudes. In such scenarios, the resulting training process should not be considered as providing valid (ϵ, δ)-DP guarantees. To guard against this, we design a randomness commitment protocol as described in Algorithm 2 that generates an unbiased randomness seed through a non-interactive process and ensures it is fixed before training starts. By hashing the full execution context–including the training identifier ID, iteration index t, batch index b_{ID}, and dataset commitment ρ_D–into the random oracle, VeriDP ensures that each randomness seed is uniquely and deterministically bound to a specific training iteration. This prevents a malicious prover from reusing, replaying, or adaptively selecting noise values across different iterations or batches. Any attempt to alter the batch contents or iteration metadata results in a mismatch in the derived seed and causes the ZKP to fail verification. As a result, batch selection and noise generation become jointly and publicly verifiable, ensuring strong non-malleability of the DP randomness across the entire training trajectory.

Algorithm 2: Unbiased Randomness Seed Commitment

Input: A random oracle $H(\cdot)$
Output: Random seed s

1: Prover sets context = (training ID, iteration index t, batch index b_{ID}, dataset commitment ρ_D), selects a uniformly random value k, and computes $r = H(\text{Com}(k) \parallel \text{context})$
2: Prover computes the seed as $s = k \oplus r$.

Adversarial model for randomness. We consider a malicious prover who controls the training execution and may attempt to bias the DP-SGD noise by choosing randomness adversarially. The prover commits to a seed witness k before proving any training iteration, and the verifier checks seed derivation inside the ZKP (cf. Fig. 1, Step (1)). This randomness adversarial model is consistent with the global threat model defined in Section 4.2.

Definition 3 (Unbiased seed generation)**.** *Let λ be the security parameter and let H be modeled as a random oracle. We say that Algorithm 2 produces an unbiased seed if, for every PPT adversary $\mathcal{A}$ controlling the prover, the derived seed $s = k \oplus H(\text{Com}(k) \parallel \text{context})$ is computationally indistinguishable from a uniformly random string U_λ, given the public transcript (including $\text{Com}(k)$ and the context string). Consequently, the Gaussian noise samples n_t generated from s using the Box–Muller transform are computationally indistinguishable from samples drawn from the target distribution $\mathcal{N}(0, \sigma^2 C^2 \mathbf{I})$ in the Random Oracle Model (ROM).*

Lemma 4.1 (Seed pseudorandomness). *Assume H is modeled as a random oracle and* Com *is a computationally hiding and binding commitment. Let $k \xleftarrow{\$} \{0,1\}^\lambda$ and define $s := k \oplus H(\text{Com}(k) \parallel \text{context})$. Then for every PPT distinguisher $\mathcal{A}$, $s \approx_c U_\lambda$, given $\text{Com}(k)$, context, and the public transcript.*

Proof sketch. By computational hiding of Com, from $\mathcal{A}$'s perspective the value k is hidden given $\text{Com}(k)$. In the ROM, $H(\text{Com}(k) \parallel \text{context})$

is distributed as a fresh uniform string in $\{0,1\}^\lambda$ (subject to negligible collision events), and is unpredictably random to $\mathcal{A}$. Finally, for any fixed (or adversarially chosen) k, the XOR of k with a uniform λ-bit string is uniform. Therefore the resulting s is computationally indistinguishable from U_λ. Binding ensures the prover cannot adaptively change k after fixing $\mathrm{Com}(k)$, ruling out adaptive biasing attempts. □

Seed Grinding Attacks and Mitigation. Seed grinding is particularly problematic in non-interactive verifiable differential privacy systems where randomness is generated solely by the prover and the verifier cannot provide externally contributed randomness. In such an attack, a malicious prover attempts to sample multiple candidate randomness seeds offline, execute training under each candidate seed, and selectively choose the seed that yields a more favorable model update (e.g., reduced effective noise magnitude or improved utility), while discarding unfavorable executions. This behavior violates the intended randomness assumption underlying differential privacy, which requires that noise be sampled independently and without adversarial bias. If unchecked, seed grinding can weaken privacy guarantees by effectively reducing the impact of injected noise. Importantly, seed grinding breaks the core independence assumption of differential privacy: the injected noise must be sampled independently of the dataset and training dynamics. By adaptively selecting favorable seeds, an adversary biases the effective noise distribution, thereby invalidating the formal (ϵ, δ)-DP guarantee. VeriDP prevents seed grinding through three complementary mechanisms:

(1) **Commit-then-derive structure.** The prover first commits to a random value k before any training proof is generated. The seed is then derived as $s = k \oplus H(\mathrm{Com}(k) \parallel \text{context})$. Since the commitment scheme is binding and H is modeled as a random oracle, the prover cannot adaptively select k after observing the derived seed. Any change in k produces a different commitment and invalidates subsequent proofs.
(2) **Context binding.** The seed derivation incorporates the full execution context, including the training identifier, iteration index, batch index, and dataset commitment ρ_D. Consequently, each seed is uniquely bound to a specific training iteration. Replaying or reusing seeds across alternative executions results in a context mismatch and proof rejection.
(3) **Proof-coupled execution.** The zero-knowledge circuit enforces that Gaussian noise generation, batch membership, and iteration metadata are all consistent with the derived seed within the same proof. Any attempt to explore multiple seeds requires recomputing a full valid proof per candidate execution, and selective disclosure is prevented by the binding commitment to k.

Mitigating residual grinding risk in deployment. In practical deployments, seed grinding can be further mitigated by (i) logging the initial commitment transcript publicly before training begins, (ii) incorporating external public randomness (e.g., a randomness beacon or block hash) into the context string, and (iii) enforcing single-shot proof generation per declared training identifier. These measures ensure that randomness remains unpredictable and non-malleable even under repeated execution attempts.

Protocol VeriDP (Iteration i)

Parameters: Clipping norm C, learning rate η, noise multiplier σ, loss function $\mathcal{L}(\cdot)$, batch size l

Dataset commitment $\rho_{\mathcal{D}}$, previous weights $\mathbf{W}_{i-1}$, prior proof π_{i-1}, commitment $\mathrm{Com}(k)$

Public Inputs: Dataset commitment $\rho_{\mathcal{D}}$, initial model commitment $\mathrm{Com}(\theta_0)$, previous weights $\mathbf{W}_{i-1}$, prior proof π_{i-1}, commitment $\mathrm{Com}(k)$

Private Witness: Mini-batch $\mathbf{B}_i = \{(x_j, y_j)\}_{j=1}^{l}$, Merkle paths $\{\pi_j^{\mathcal{D}}\}_{j=1}^{l}$, seed witness k, per-example gradients $\{\mathbf{g}_j\}_{j=1}^{l}$

Outputs: Updated weights $\mathbf{W}_i$, recursive proof π_i

Protocol:

(1) **Seed correctness.** Prover proves knowledge of k such that $s = k \oplus H(\mathrm{Com}(k))$ (cf. Algorithm 2). Prover additionally proves that the public inputs $(t, b_{\mathrm{ID}}, \rho_D)$ are correctly encoded into the hash input such that $s = k \oplus H(\mathrm{Com}(k) \| \text{training ID} \| t \| b_{\mathrm{ID}} \| \rho_D)$.
Unbiasedness of the derived seed follows from Lemma 4.1.
(2) **Dataset membership enforcement.** For each sample $j = 1, \ldots, l$, prover provides Merkle proof $\pi_j^{\mathcal{D}}$ and proves: $\mathrm{Verify}(\rho_{\mathcal{D}}, (x_j, y_j), \pi_j^{\mathcal{D}}) = 1$
(3) **Gradient correctness.** Prover computes and proves: $\mathbf{g}_j = \nabla_{\mathbf{W}_{i-1}} \mathcal{L}(\mathbf{W}_{i-1}, (x_j, y_j))$
(4) **Clipping correctness.** Prover computes: $\hat{\mathbf{g}}_j = \mathbf{g}_j \cdot \min\left(1, \frac{C}{\|\mathbf{g}_j\|_2}\right)$ and proves correctness.
(5) **Averaging correctness.** Prover computes: $\hat{\bar{\mathbf{g}}} = \frac{1}{l} \sum_{j=1}^{l} \hat{\mathbf{g}}_j$ and proves correctness.
(6) **Verifiable noise generation.** Using seed s, prover generates Gaussian noise $\mathbf{n} \sim \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})$ inside the circuit via Box–Muller (Fig. 2) and proves correctness.
(7) **Weight update.** Prover proves: $\mathbf{W}_i = \mathbf{W}_{i-1} - \eta(\hat{\bar{\mathbf{g}}} + \mathbf{n})$
(8) **Recursive enforcement.** Prover generates recursive proof: $\pi_i = \mathrm{IVC.Prove}(\rho_{\mathcal{D}}, \mathrm{Com}(\theta_0), \mathbf{W}_i, \pi_{i-1})$

Figure 1: VeriDP iteration with dataset binding and recursion. Mini-batch elements are verified against $\rho_{\mathcal{D}}$ via Merkle proofs, and the commitment and prior proof π_{i-1} are carried in the IVC state to ensure consistency and DP-SGD correctness across iterations.

Remark 1. ***Preventing grinding and selective bias.*** *The seed is bound to the full execution context (training ID, iteration index, batch index, and dataset commitment). Thus, replaying* $\mathrm{Com}(k)$ *across different iterations changes the hash input and yields a different seed. Moreover, since* $\mathrm{Com}(k)$ *is fixed before proving the DP-SGD iteration, the prover cannot adaptively search for a favorable seed without being detected.*

4.3.3 Zero-Knowledge Proof of Differential Privacy Guarantees. We construct a ZKP system for verifying that a DP-SGD update was computed with the correct parameters (C, η, σ) and Gaussian noise, without revealing the training data, gradients, or randomness. The prover first commits to a random value k and derives the seed $s = k \oplus H(\mathrm{Com}(k))$ (cf. Algorithm 2), ensuring unbiased randomness. Using s, Gaussian noise is deterministically generated inside the

Input: Seeds s_1, s_2 (cf. Algorithm 2), constants σ, C
Output: Noise values n_1, n_2 such that $n_i \sim \mathcal{N}(0, (\sigma C)^2)$

Circuit steps:

- **PRNG initialization:** Use the seeds s_1 and s_2 to deterministically generate two uniform values $U_1, U_2 \in (0, 1)$, respectively. This can be implemented using a secure hash function (e.g., SHA-256) or a sponge construction (e.g., Poseidon, Rescue).
- **Convert U_1, U_2 to fixed-point representation:** Convert the floating-point $(0, 1)$ values into fixed-point format with a specified precision (e.g., Q16.16).
- **Box-Muller transform:**
  - Compute $L = -2 \cdot \log(U_1)$ using a Chebyshev or piecewise approximation of log.
  - Compute $R = \sqrt{L}$ using a polynomial approximation of $\sqrt{\cdot}$.
  - Compute $\theta = 2\pi U_2$ using constant multiplication and modular reduction.
  - Compute $\cos(\theta)$ and $\sin(\theta)$ using either:
    - Chebyshev polynomial approximations, or
    - Precomputed LUTs and interpolation within the circuit.
  - Compute $z_1 = R \cdot \cos(\theta)$ and $z_2 = R \cdot \sin(\theta)$ via fixed-point multiplication.
- **Noise scaling:**
  - Compute $n_1 = \sigma C z_1$ and $n_2 = \sigma C z_2$ using fixed-point multiplication.

Figure 2: ZK Circuit for Gaussian Noise via Box-Muller [7]

circuit via the verifiable Box–Muller transform (cf. Fig. 2). This involves: (i) generating two uniform values $U_1, U_2 \in (0, 1)$ from the seed using a cryptographically secure PRNG, (ii) converting these values to fixed-point representation, (iii) computing $L = -2\log(U_1)$ and $R = \sqrt{L}$ using polynomial or piecewise approximations for log and $\sqrt{\cdot}$, (iv) computing $\theta = 2\pi U_2$ and evaluating $\cos(\theta)$ and $\sin(\theta)$ either via Chebyshev approximations or precomputed LUTs, and (v) producing z_1 and z_2 as standard Gaussian samples and scaling them by σC. All of these operations are performed within the circuit to ensure complete verifiability.

We express all the above computations as arithmetic circuits, and the prover proves the correctness of the circuit evaluation. For linear operations, the parties execute sublinear sumcheck protocols [3, 25, 26, 46], whereas for non-linear operations, they use a generic GKR-based proof system, such as Virgo++ [52] in our implementation. VeriDP, described in Fig. 1, integrates these components: the prover proves correct gradient computation, clipping, averaging, and noise generation, and that the final model weights satisfy the DP-SGD update rule. This achieves end-to-end verifiability of the claimed differential privacy guarantees while keeping both the data and the randomness private. As shown in Fig. 1 (Step 2 and Step 8), every mini-batch element is accompanied by a Merkle membership proof against the committed dataset $\rho_{\mathcal{D}}$, and the dataset commitment is carried forward as part of the recursive IVC state, ensuring dataset consistency across all training iterations. The batch index b_{ID} is treated as part of the public IVC state and is included in the randomness derivation context. The prover must show that all Merkle membership proofs in Step (2) correspond to the same b_{ID} that is hashed into the seed generation, cryptographically binding batch selection and noise generation to a single, verifiable training iteration.

Implementation of Non-Linear Operations. Non-linear operations (e.g., ReLU, min, square root, logarithm, cosine) are implemented using low-degree polynomial approximations.

- *ReLU.* $\text{ReLU}(x)$ is approximated using a degree-d polynomial $P_{\text{ReLU}}(x)$ constructed via Chebyshev approximation over a bounded interval $[-B, B]$. The approximation error satisfies:
$$\sup_{x \in [-B,B]} |\text{ReLU}(x) - P_{\text{ReLU}}(x)| \leq \epsilon_{\text{ReLU}}.$$
- *Gradient Clipping.* Gradient clipping enforces $\mathbf{g}_t \leftarrow \mathbf{g}_t \cdot \min\left(1, \frac{C}{\|\mathbf{g}_t\|_2}\right)$. The min operation is implemented using $\min(a, b) = \frac{a+b-|a-b|}{2}$. Absolute value is approximated using a low-degree polynomial over the bounded domain induced by clipping parameter C.
- *Error Control.* All approximation errors are bounded and absorbed into the declared fixed-point precision parameter f, which is part of the public input and enforced inside the circuit.

Remark 2. ***Separation of Execution and Verification.*** *VeriDP distinguishes between the arithmetic used for model training and the arithmetic enforced inside the zero-knowledge circuit. The training computation itself may be carried out using standard floating-point arithmetic in existing machine learning frameworks. The role of the circuit is to verify that the committed training values admit a consistent fixed-point representation and that the injected noise is consistent with a verifiable Gaussian sampler parameterized by (σ, C, f). Consequently, VeriDP does not modify the optimization dynamics of DP-SGD and does not introduce additional noise beyond that required by the privacy mechanism. The verification layer enforces correctness and privacy compliance without constraining the numerical format used by the underlying training system.*

4.3.4 Numerical Accuracy of Box–Muller Sampling and Impact on DP. VeriDP generates Gaussian noise inside the circuit using a fixed-point implementation of the Box–Muller transform. This requires (i) finite-precision arithmetic with public fixed-point precision parameter f, and (ii) polynomial, piecewise, or LUT-based approximations of nonlinear functions such as $\log(\cdot)$, $\sqrt{\cdot}$, $\sin(\cdot)$, and $\cos(\cdot)$. We analyze how these numerical effects influence the resulting noise distribution and the claimed DP guarantees.

Approximation model. Let $\mathcal{N}(0, \sigma^2 C^2)$ denote the ideal continuous Gaussian distribution required by DP-SGD. Let $\widetilde{\mathcal{N}}$ denote the (generally discrete) distribution induced by the in-circuit fixed-point Box–Muller implementation with precision f. We model each produced noise coordinate as $\tilde{n} = n + e$, where $n \sim \mathcal{N}(0, \sigma^2 C^2)$ and e captures fixed-point rounding and approximation error. For a fixed choice of approximation scheme and precision f, the circuit enforces a deterministic bound

$$|e| \leq \Delta_f, \tag{1}$$

where Δ_f decreases with f (e.g., $\Delta_f = O(2^{-f})$ plus the chosen approximation error).

Distance to the ideal Gaussian. The Box–Muller transform is smooth on its effective domain (with U_1 clamped away from 0 inside the circuit), and thus locally Lipschitz. Under bounded arithmetic error Δ_f, the output perturbation remains bounded, implying that the induced distribution remains close to the ideal Gaussian. For two probability distributions P and Q over a common

domain Ω, we denote by $\mathrm{TV}(P,Q) := \sup_{S\subseteq\Omega} |P(S) - Q(S)|$ their total variation distance. their total variation distance. In particular, $\mathrm{TV}\left(\widetilde{\mathcal{N}}, \mathcal{N}(0,\sigma^2C^2)\right) \le \tau_f$, for some $\tau_f = O(\Delta_f)$ determined by the fixed-point precision and approximation scheme.

Effect on differential privacy. Differential privacy is robust to small perturbations in the underlying randomness distribution. If the ideal mechanism $\mathcal{M}$ is (ε,δ)-DP and the implemented mechanism $\widetilde{\mathcal{M}}$ differs from $\mathcal{M}$ by at most τ_f in total variation distance (over the noise distribution), then $\widetilde{\mathcal{M}}$ satisfies $(\varepsilon, \delta + \tau_f)$-DP. Thus, numerical approximation can be accounted for as an additive increase in δ, while ε remains unchanged. By selecting f such that τ_f is negligible (and in particular $\tau_f \ll \delta$), the declared privacy guarantee is preserved.

Effect on RDP accounting. Our privacy accounting uses RDP composition. The RDP divergence depends continuously on the noise distribution parameters. Since the implemented distribution is within τ_f of the ideal Gaussian in total variation distance, the induced per-iteration RDP parameter changes by at most $O(\tau_f)$, and the cumulative deviation over T iterations is bounded by $O(T\cdot\tau_f)$. In our setting, f is a public parameter enforced by the circuit and is chosen such that $T\cdot\tau_f$ remains negligible relative to the target δ in the final (ε,δ) conversion.

Impact on model performance. In addition to the privacy guarantee, we clarify how the fixed-point and approximation error in the in-circuit Box–Muller sampler can affect the training dynamics and final utility. Recall that each noise coordinate produced by the circuit can be written as $\tilde{n} = n + e$, where $n \sim \mathcal{N}(0,\sigma^2C^2)$ and the arithmetic/approximation error satisfies the deterministic bound $|e| \le \Delta_f$ enforced by the circuit (cf. Equation 1). Therefore, the implemented DP-SGD update is identical to the ideal DP-SGD update, except for an additional *bounded* perturbation term on the noisy gradient $\tilde{g}_t = \bar{\tilde{g}}_t + n_t + e_t$, $\|e_t\|_2 \le \sqrt{d}\,\Delta_f$, where d is the gradient dimension. For standard DP-SGD regimes, the injected Gaussian noise scale σC dominates the discretization/approximation scale Δ_f (e.g., for $f \in \{16, 32\}$ fractional bits, $\Delta_f = 2^{-f}$). Consequently, the additional perturbation behaves like a vanishingly small extra noise source and does not materially change the optimization trajectory. More formally, if the loss is L-smooth (i.e., $\nabla\ell(\theta)$ is L-Lipschitz) and DP-SGD uses step size η, then the one-iteration deviation between the ideal iterate θ_{t+1} and the implemented iterate $\tilde{\theta}_{t+1}$ satisfies $\|\tilde{\theta}_{t+1} - \theta_{t+1}\|_2 = \eta\|e_t\|_2 \le \eta\sqrt{d}\,\Delta_f$. By iterating this bound (and using standard stability arguments for smooth objectives), the cumulative deviation over T iterations grows at most linearly in T: $\|\tilde{\theta}_T - \theta_T\|_2 \le \eta T\sqrt{d}\,\Delta_f$, which is negligible for the precisions we use in practice. Thus, the numerical effects of the verifiable Box–Muller sampler can be made arbitrarily small by choosing the public precision parameter f, without changing the DP mechanism itself.

4.4 Comparison with Confidential-DPproof

Confidential-DPproof [43] is the closest prior work to our construction. Both works aim to replace empirical privacy auditing with cryptographic enforcement. However, the two systems differ fundamentally in structure, verifiability guarantees, and composability.

Monolithic vs. Incremental Proof Structure. Confidential-DPproof encodes the entire DP-SGD training run as one monolithic circuit and outputs a single proof of privacy compliance. As a result, the prover's work, proof size, and verifier cost grow linearly with the number of iterations. In contrast, VeriDP models each DP-SGD iteration as its own verifiable circuit and composes these proofs recursively using IVC. This design ensures that proof size and verification time remain constant regardless of training length. This asymptotic scalability is not achievable with a monolithic circuit.

Granularity of Privacy Enforcement. [43] verifies only a global (ϵ,δ) guarantee for the entire training run. Intermediate iterations are not individually auditable, and a corrupt participant could violate clipping or noise addition during intermediate updates without detection, as long as the final circuit is satisfied. VeriDP enforces per-iteration correctness: clipping, averaging, noise generation, and parameter updates are all individually proven. This provides continuous enforcement of differential privacy and stronger security in adversarial or distributed settings.

Randomness Commitment. Confidential-DPproof uses an interactive seed generation protocol requiring an online auditor. If the auditor is replaced by a deterministic Fiat–Shamir transform [15], the scheme becomes non-interactive but loses its interactive randomness-quality guarantees. VeriDP derives the randomness seed from a prover-committed value using a deterministic, non-interactive transform, ensuring unbiased, publicly verifiable randomness without requiring an online auditor.

Commitment Model and Public Verifiability. [43] employs IT-MAC commitments, which are not hiding and require a shared key with an auditor. They provide private verification only. VeriDP uses polynomial commitments, which are hiding, binding, and publicly verifiable, and integrate naturally with recursive proofs. This enables decentralized auditing and long-term verifiability–capabilities that IT-MAC–based systems cannot support.

Verifiable Privacy Accounting. Confidential-DPproof provides a certificate for a single (ϵ,δ) value corresponding to the full training run, but it neither implements verifiable intermediate privacy accounting nor supports more advanced notions such as [33]. As a result, auditors must trust the prover's claim about the privacy budget used to generate the final certificate. In contrast, VeriDP integrates tight RDP-based privacy accounting directly into the proof system: each iteration certifies the exact per-iteration contribution implied by the clipped gradients and the verifiably generated Gaussian noise, and the composed RDP curve is enforced across all iterations. The privacy accountant is therefore cryptographically tied to the iteration-level constraints, enabling transparent and fine-grained auditing of the evolving privacy loss. This level of verifiable, iteration-level privacy accounting is not supported by Confidential-DPproof and is critical for long training runs, adaptive optimizers, or federated deployments where small deviations can accumulate into a significant privacy parameter.

In general, even if Confidential-DPproof were modified to use Fiat–Shamir randomness and polynomial commitments, it would still lack the recursive proof structure necessary for per-iteration verifiable DP-SGD. The architectural advances in VeriDP–namely IVC-based scalability, per-iteration privacy enforcement and public composability–represent fundamental capabilities beyond what monolithic designs can provide.

Remark 3. *Unlike monolithic constructions, VeriDP anchors the entire recursive proof chain to a publicly committed initial model state, ensuring that all verified updates are cryptographically bound to a unique training origin.*

4.5 Security Analysis

Theorem 4.2 (Security of VeriDP)**.** *Let $R_{DP\text{-}SGD}$ be the NP relation from Definition 2. Let Π_{VeriDP} be the protocol in Fig. 1 instantiated with (i) a binding/hiding dataset commitment, (ii) the seed-derivation procedure $s = k \oplus H(\mathrm{Com}(k))$ of Algorithm 2, and (iii) a zero-knowledge argument system (sumcheck/GKR with polynomial commitments) to enforce the per-iteration constraints of Algorithm 1, including verifiable Gaussian noise via the Box-Muller circuit of Fig. 2. Soundness additionally guarantees that any accepting proof implies the existence of a unique initial model θ_0 consistent with the public commitment $\mathrm{Com}(\theta_0)$, from which all subsequent model states θ_t are derived via valid DP-SGD transitions. Then Π_{VeriDP} is a Zero-Knowledge Proof of DP-SGD for $R_{DP\text{-}SGD}$ in the sense of Definition 2.*

We refer the reader to Appendix B for a detailed proof.

4.5.1 Numerical Precision and Security Analysis. Prior works [12, 21] have demonstrated that differential privacy implementations based on floating-point arithmetic may be vulnerable to precision-based side-channel attacks, including timing leakage and hardware-dependent rounding behavior. These attacks exploit implicit numerical behavior that is not captured by the formal privacy mechanism but emerges from low-level execution semantics. In contrast, VeriDP makes numerical precision an explicit, public, and cryptographically enforced component of the protocol. All Gaussian noise samples are generated within the zero-knowledge circuit using fixed-point arithmetic with a declared fractional precision parameter f. The verifier checks that all arithmetic operations, including logarithms, square roots, trigonometric approximations, and scaling by σC, are consistent with this fixed-point representation. As a result, the noise distribution is no longer an implicit artifact of floating-point hardware or compiler-level optimizations, but rather a formally specified and auditable distribution that is uniquely determined by the committed randomness seed and the public precision parameter. This design eliminates hidden rounding channels and prevents adversaries from exploiting implementation-dependent numerical behavior.

Distributional Closeness Under Fixed-Point Noise. Let $n \sim \mathcal{N}(0, \sigma^2 C^2 I_d)$ denote an ideal Gaussian noise vector in d dimensions, and let $\tilde{n}$ be the noise vector generated by the fixed-point Box–Muller circuit in VeriDP using f fractional bits. Let $\Delta = 2^{-f}$ denote the maximum representable discretization unit.

Lemma 4.3. *The expected relative error between $\tilde{n}$ and n satisfies $\frac{\mathbb{E}[\|\tilde{n}-n\|_2]}{\mathbb{E}[\|n\|_2]} \le \frac{\sqrt{d}\cdot\Delta}{\sigma C}$.*

Sketch. Each arithmetic operation in the fixed-point Box–Muller circuit introduces a bounded additive error of at most Δ per coordinate due to rounding. By linearity of expectation and independence across dimensions, the total perturbation accumulates as $\|\tilde{n} - n\|_2 \le \sqrt{d} \cdot \Delta$. Meanwhile, $\mathbb{E}[\|n\|_2] = \Theta(\sigma C\sqrt{d})$, yielding the stated bound. □

For typical DP-SGD settings with $\sigma \ge 1$ and $f \in \{16, 32\}$, this ratio is negligible (e.g., below 10^{-4} for $d \le 10^6$), implying that discretization error is dominated by the magnitude of the Gaussian noise required for privacy.

4.6 Application of VeriDP in Federated Learning

We formalize VeriDP for federated learning, specifying the training protocol, recursive proof structure, and privacy accounting (consistent with Definition 2 and Protocol VeriDP).

Federated Learning Setting. We consider a federated training process consisting of i) K clients indexed by $k \in \{1, \dots, K\}$, ii) R communication rounds, iii) client-local datasets D_k with dataset commitments ρ_{D_k}, and iv) a central aggregator maintaining global model parameters $\mathbf{W}_r$ at round r. At each communication round r:

(1) The aggregator broadcasts the current global model $\mathbf{W}_r$.
(2) Each selected client k initializes $\mathbf{W}_r^{k,0} := \mathbf{W}_r$.
(3) Client k performs E local DP-SGD iterations using Protocol VeriDP. A local iteration refers to a standard DP-SGD iteration executed by a client during a federated round; thus, the term iteration used throughout the paper and local iteration in the FL context denote the same DP-SGD update primitive.
(4) The client outputs updated model $\mathbf{W}_r^{k,E}$ and recursive proof π_r^k.
(5) The aggregator verifies π_r^k before incorporating the update.
(6) Verified updates are aggregated as $\mathbf{W}_{r+1} = \sum_{k\in\mathcal{S}_r} \frac{|D_k|}{\sum_{j\in\mathcal{S}_r}|D_j|} \mathbf{W}_r^{k,E}$, where $\mathcal{S}_r$ denotes participating clients.

Only updates accompanied by valid proofs are included in the aggregation phase.

Number of Local DP-SGD Iterations. Each client performs exactly E local DP-SGD iterations per communication round. The value E is fixed prior to training and is treated as a public hyperparameter included in the recursive IVC state. Let $t \in \{1, \dots, E\}$ denote the local iteration index within round r. For each local iteration:

$$\tilde{\mathbf{g}}_{r,t} = \frac{1}{b} \sum_{i\in B_{r,t}} \hat{\mathbf{g}}_i + \mathbf{n}_{r,t}, \quad \hat{\mathbf{g}}_i = \frac{\mathbf{g}_i}{\max\left(1, \frac{\|\mathbf{g}_i\|_2}{C}\right)}, \quad \mathbf{n}_{r,t} \sim \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}),$$

and the model update is $\mathbf{W}_r^{k,t} = \mathbf{W}_r^{k,t-1} - \eta\tilde{\mathbf{g}}_{r,t}$. Across the entire training process, each client executes $T = R \cdot E$ DP-SGD iterations.

Recursive Proof Generation Across Local Iterations. Within each client, VeriDP applies IVC to generate a single succinct proof covering all E local iterations. At local iteration t, the prover computes: $\mathbf{W}_r^{k,t} = \mathbf{W}_r^{k,t-1} - \eta\tilde{\mathbf{g}}_{r,t}$, and produces recursive proof:

$$\pi_r^{k,t} = \mathsf{IVC.Prove}\big(\rho_{D_k}, \mathrm{Com}(\boldsymbol{\theta}_0), \mathbf{W}_r^{k,t}, \pi_r^{k,t-1}\big).$$

The base case binds the proof to $\mathbf{W}_r^{k,0} = \mathbf{W}_r$. After E local iterations, the client outputs $\pi_r^k := \pi_r^{k,E}$. By recursion, proof size and verifier time are independent of E, while prover time scales linearly in E; hence the aggregator checks a single constant-size proof even though E local DP-SGD steps are executed.

Proof Guarantees in the Federated Setting. For each round r, client k proves that for all local steps $t \in \{1, \dots, E\}$:

(1) All mini-batch elements belong to committed dataset ρ_{D_k} via Merkle proofs.

(2) Per-example gradients $\mathbf{g}_i$ were correctly computed.
(3) Clipping was performed using public norm bound C.
(4) Gaussian noise $\mathbf{n}_{r,t}$ was generated as $\mathbf{n}_{r,t} \sim \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})$, using a context-bound seed $_{r,t} = H(\text{training ID} \| r \| t \| \rho_{D_k})$.
(5) Model updates satisfy $\mathbf{W}_r^{k,t} = \mathbf{W}_r^{k,t-1} - \eta \tilde{\mathbf{g}}_{r,t}$.

This provides per-round, per-client cryptographic enforcement of the DP-SGD protocol.

5 Evaluation

We implemented VeriDP and report experiments instantiated over a 255-bit prime field (≈128-bit security), modeling hashes as SHA-256 in the ROM and using discrete-log-based polynomial commitments over 128-bit-secure elliptic-curve groups.

5.1 System Implementation

5.1.1 Software. We implemented the scheme in C++, resulting in roughly 5,000 lines of code. The complete implementation is available at [1]. Some of our algorithms on the sumcheck protocol and the GKR protocol are based on the open-source implementation of [3, 25, 26, 33]. We use polynomial commitment schemes for efficient proof generation because of their good prover time and reasonable proof size for the witness in our experiments. The security of the scheme relies on the discrete-log assumption. The prover time is $O(N)$ and the proof size and the verifier time are $O(\sqrt{N})$ for a polynomial of size N. We use a 61-bit Mersenne prime field ($2^{61} - 1$) as the base field for field arithmetic, which offers efficient FFT operations [37]. For applications requiring higher security guarantees, we provide an optional 256-bit BLS12-381 scalar field implementation using the `mcl` library [18], as the 61-bit field may not provide sufficient security for all deployment scenarios. The BLS12-381 field offers 128-bits of security and is FFT-friendly with a large power-of-2 root of unity. For neural network computations, we use LibTorch (PyTorch C++ API) to interface with deep learning operations. We bridge floating-point tensor computations to field arithmetic using Q24.24 fixed-point quantization, ensuring exact arithmetic matching between the prover's computations and the verifier's checks. We implement IVC for aggregating proofs across training batches, reducing the final proof size to constant rather than linear in the number of batches.

5.1.2 Dataset. We evaluate VeriDP on:

(1) **MNIST** handwritten digit classification dataset [54], which consists of 60,000 training images and 10,000 test images of 28×28 grayscale digits. Each image is normalized with a mean of 0.1307 and a standard deviation of 0.3081.
(2) **CIFAR-10** natural image classification dataset [24], which consists of 50,000 training images and 10,000 test images of 32×32 RGB images across 10 classes. Each image is normalized channel-wise with mean $(0.4914, 0.4822, 0.4465)$ and standard deviation $(0.2023, 0.1994, 0.2010)$.

The dataset is committed to a Merkle tree before training begins, with each sample (image pixels and label) hashed using a collision-resistant hash function. This commitment ensures that the prover cannot manipulate the training data after the commitment is published. For our experiments, we use subsets of the training data ranging from 32 to 16,000 samples to evaluate scalability by varying batch sizes of samples.

5.1.3 Model. We evaluate three neural network architectures of varying complexity:

(1) **MLP**: A two-layer fully connected network with 101,770 trainable parameters. The architecture consists of an input layer (784 neurons, corresponding to the flattened 28 × 28 image), a hidden layer with 128 neurons and ReLU activation, and an output layer with 10 neurons for digit classification. The parameter breakdown is: fc1 ($784128 + 128 = 100,480$ parameters) and fc2 ($12810 + 10 = 1,290$ parameters).
(2) **CNN**: A four-layer convolutional neural network with 421,642 trainable parameters. The architecture consists of two convolutional layers (conv1: 1→32 channels with 3×3 kernels, conv2: $32 \to 64$ channels with 3×3 kernels), each followed by ReLU activation and 2×2 max pooling, and two fully connected layers (fc1: $3136 \to 128$, fc2: $128 \to 10$). The parameter breakdown is: conv1 (320 parameters), conv2 (18,496 parameters), fc1 (401,536 parameters), and fc2 (1,290 parameters).
(3) **ResNet18**: A residual convolutional network with 11,173,962 trainable parameters. The model starts with a 3×3 stem convolution ($3 \to 64$), followed by four residual stages with channel widths 64, 128, 256, 512, using 2 BasicBlocks per stage. Each BasicBlock has two 3×3 convolutions with a skip connection and when spatial size/channel count changes, a 1×1 downsampling projection is used on the shortcut path. The head is global average pooling and a fully connected layer ($512 \to 10$).

5.1.4 Hardware. We run all of the experiments on a machine with 12^{th} Generation Intel(R) Core i7-12650H (2.30 GHz) and 64GB of RAM. Our current implementation is not parallelized, and we only utilize a single CPU core. The large memory is used to run experiments on neural networks with varying training batches. On one hand, the memory usage is actually the bottleneck to further scale veriDP, and it is an interesting future work to improve it. We report the average running time of 10 executions.

5.2 Performance

We evaluate VeriDP on several dimensions: (1) prover and verifier efficiency, (2) scalability with respect to model size and training samples, (3) proof size, and (4) memory usage.

Implementation of Non-Linear Operations We describe the implementation of non-linear operations used in the arithmetic circuits underlying the verifiable DP-SGD construction.

- The ReLU activation function is implemented via algebraic constraints that encode its piecewise definition directly in the field. This avoids the use of polynomial approximation and therefore introduces no approximation error while remaining compatible with zero-knowledge verification.
- The log and $\sqrt{\cdot}$ functions arise in the Box–Muller transform used for Gaussian noise generation. Both functions are approximated over the underlying finite field using Chebyshev polynomials. We employ a degree-5 Chebyshev approximation for log and a degree-4 Chebyshev approximation for $\sqrt{\cdot}$. These degrees were selected to balance numerical accuracy and arithmetic circuit complexity under the declared fixed-point precision.

Table 1: MLP vs CNN vs ResNet Model Comparison

		MLP (MNIST)	CNN (MNIST)	ResNet (CIFAR-10)
Parameters	Clipping Norm	10	10	10
	Noise Multiplier	1.1	1.1	1.1
	Learning Rate	0.01	0.01	0.01
	Batches	20	20	20
Proof (KB)	Final Proof Size	3.93	4.08	4.08
	Average Batch Proof Size	3.20	3.50	4.13
Prover (s)	Total Prover Time	53.25	217.51	5162.54
	Clipping Time	9.27	37.22	903.70
	Noise Generation Time	1.51	5.91	184.66
	Weight Update Time	20.85	90.04	2890.96
Verifier (ms)	Verifier Time	3.21	2.38	4.96
Privacy Budget ϵ		2.92	2.92	2.92

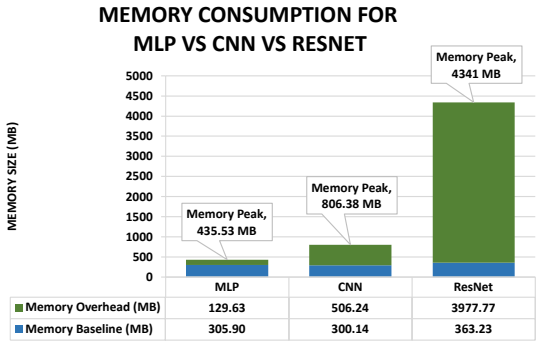

Figure 3: Memory Consumption for MLP vs CNN vs ResNet models

- The trigonometric functions required by the Box–Muller transform are implemented using a lookup table (LUT) with 256 uniformly spaced entries over the interval $[0, 2\pi)$. The circuit enforces correct index selection and value retrieval, ensuring efficient and verifiable evaluation within the ZKP system.

Overall, these design choices reduce circuit depth while preserving numerical stability and consistency with the fixed-point arithmetic constraints enforced by the verification circuit.

5.2.1 MLP vs CNN vs ResNet Comparison. We first compare the performance of VeriDP on three neural network architectures: MLP (101,770 parameters), CNN (421,642 parameters) and ResNet (11,173,962 parameters). All models are trained with identical DP-SGD parameters to enable fair comparison, as shown in Table 1. The CNN requires approximately 4× longer prover time than the MLP, which corresponds directly to the ratio of model parameters ($421,642/101,770 \approx 4.1\times$). Including ResNet, the trend is even clearer; compared with MLP and CNN, ResNet18 incurs a much larger prover cost with prover time around 86 minutes, which is broadly consistent with parameter scaling (ResNet/CNN parameter ratio $\approx 26\times$, prover-time ratio $\approx 24\times$). All models achieve the same ϵ differential privacy for identical DP-SGD parameters, as expected. The privacy budget depends only on the noise multiplier, sampling rate, and number of iterations rather than the model architecture. Despite the differences in model size, all models achieve similar verifier times (2-5 ms) and final proof sizes (3.9-4.1 KB), demonstrating that verification complexity is independent of model size. Based on Fig. 3, we can also see that MLP incurs approximately 130 MB overhead above the 300 MB baseline, while CNN and ResNet incur approximately 506 MB and 3978 MB overhead respectively. The higher memory usage for CNN and ResNet is due to tracking more intermediate values for the larger models.

5.2.2 Scalability Analysis. VeriDP generates 4 proofs per batch (one each for gradient clipping, gradient averaging, noise addition, and weight update). Based on the performance results by varying batch sizes on CNN model (MNIST) and ResNet model (CIFAR-10) as displayed in Table 2 and Table 3 respectively, the prover time scales linearly with the number of training samples, confirming the $O(N)$ complexity of our proof system. The linear relationship demonstrates that VeriDP can scale to longer training runs with predictable overhead. With CNN, the prover time per sample remains stable at approximately $342 - 357ms$ across the configurations whereas with ResNet, the prover time per sample takes between $8.30 - 8.50s$, hence demonstrating predictable performance regardless of training duration. In addition, by examining how each component of the prover time behaves across different batch sizes in all models, we observe that the major contributors, i.e, clip norm, noise generation, noise addition, weight update, and others[1], scale proportionally with the total prover time. We notice that their relative contributions remain fairly consistent as the batch size increases. To illustrate this, we present a percentage stacked column graph that summarizes each component's share, computed by averaging their values across all experimental runs, in Fig. 4. Another critical property of VeriDP is that verification remains efficient regardless of training duration. We analyze the verifier time and proof size across different configurations. We noticed that the verifier generally completes in $2-6ms$. This $O(\sqrt{N})$ complexity enables practical third-party verification. Through IVC aggregation, the proof size remains constant at approximately 3.5-4 KB per batch, independent of training duration. Multiple batch proofs are aggregated into a final succinct proof. These results demonstrate that verifiable differential privacy is practical for real-world machine learning workloads, with verification costs that are negligible compared to proof generation.

5.2.3 Comparison with DPproof [43]. To evaluate practical proving efficiency, we compare VeriDP (yellow and green lines) against DPproof (red and blue lines) across linear and logistic regression while varying both dataset size and batch size in Fig. 5. We noticed that with small batches (for e.g, 2), VeriDP can be occasionally slower than DPproof because fixed protocol overheads (setup, transcript/proof orchestration, and aggregation) dominate runtime. As batch size increases, however, VeriDP consistently outperforms DPproof, and the gap widens for larger datasets. This suggests VeriDP amortizes fixed costs better and scales more smoothly in batched proving, reducing prover time per sample at moderate-to-large batch sizes, while DPproof shows steeper growth and larger runtime spikes. An important caveat is that DPproof's benchmarks are restricted to linear and logistic regression. VeriDP is evaluated

[1]Remaining Prover time with Merkle proofs, forward/backward pass, object construction and logging as the largest contributors.

Table 2: VeriDP's Performance with CNN model on MNIST dataset for Different Batch Sizes

Batch Size	Final Proof Size (KB)	Verifier Time(ms)	Total Prover Time (s)	Clip Norm (s)	Noise Generation (s)	Noise Addition (s)	Weight Update (s)	Others (s)
1	3.61	4.36	11.40	1.87	0.31	0.31	4.53	4.38
2	4.08	2.77	22.53	3.74	0.63	0.63	9.15	8.38
5	4.08	2.70	55.03	9.36	1.50	1.50	22.55	20.13
10	3.61	2.65	109.20	18.67	2.95	2.95	45.09	39.53
20	4.08	2.69	218.35	37.29	5.95	5.95	90.29	78.87
50	4.08	2.81	544.25	94.31	14.71	14.71	225.68	195.82
100	3.61	8.8	1093.46	187.97	29.78	29.78	454.29	391.58
200	4.08	2.74	2196.02	379.27	59.48	59.48	910.44	787.35
500	4.08	4.39	5256.95	903.64	141.75	141.75	2180.74	1888.71

Table 3: VeriDP's Performance with ResNet model on CIFAR-10 dataset for Different Batch Sizes

Batch Size	Final Proof Size (KB)	Verifier Time(ms)	Total Prover Time (s)	Clip Norm (s)	Noise Generation (s)	Noise Addition (s)	Weight Update (s)	Others (s)
1	3.61	6.64	269.21	52.56	9.23	9.23	145.56	52.62
2	4.08	5.26	531.45	92.63	19.04	19.04	297.08	103.64
5	4.08	5.11	1331.91	233.03	47.51	47.51	745.49	258.30
10	3.61	5.26	2638.11	461.38	94.47	94.47	1476.00	511.62
20	4.08	4.35	5162.54	903.70	184.66	184.66	2890.96	998.21
50	4.08	6.39	12838.96	2241.29	456.87	456.87	7201.79	2481.24
100	3.61	30.07	26627.92	4686.59	915.45	915.45	14962.02	5146.61

on these classical models and additionally demonstrated on neural architectures (MLP, CNN, and ResNet). Therefore, the overlap comparison shows competitive-to-superior performance on shared tasks, and VeriDP further generalizes from low-complexity convex models to practical deep-learning workloads.

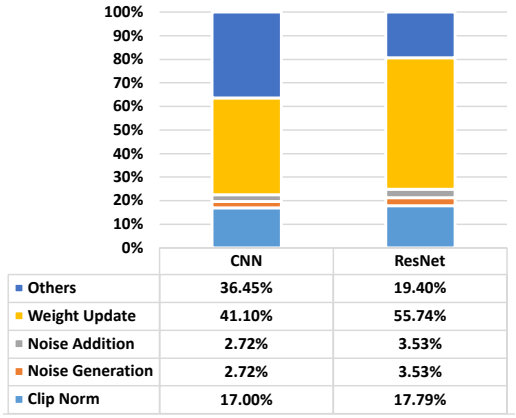

Figure 4: Prover Time Distribution across Components in CNN and ResNet

6 Discussion and Limitations

VeriDP enables verifiable DP-SGD by producing a proof per training iteration and linking them via IVC, yielding continuous privacy enforcement and enabling verifiable FL and transparent audit logs. While IVC gives constant-size proofs, per-iteration proving overhead and arithmetic/circuit constraints remain significant (and privacy accounting/hyperparameter choice is not certified), but the architecture still delivers scalable per-step verification, public

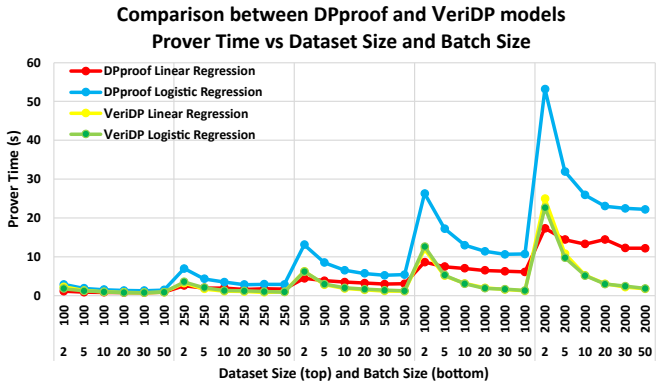

Figure 5: Comparing Prover Time across DPproof and VeriDP models

composability, and decentralized auditability beyond monolithic approaches like Confidential-DPproof.

Acknowledgments

Amir R. Asadi is supported by Leverhulme Trust grant ECF-2023-189 and Isaac Newton Trust grant 23.08(b). The authors from the Barkhausen Institut have been funded by the German Federal Ministry of Research, Technology, and Space through the SEMECO project (Grant No. 03ZU1210AA). They also receive financial support from the budget adopted by the Saxon State Parliament in Germany.

References

[1] 2026. VeriDP: Verifiable DP-SGD Framework. https://github.com/Barkhausen-Institut/VeriDP.
[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In

Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 308–318.

[3] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. 2024. Zero-knowledge proofs of training for deep neural networks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 4316–4330.

[4] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Beguelin. 2012. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 97–110.

[5] Ari Biswas and Graham Cormode. 2022. Verifiable differential privacy. *arXiv preprint arXiv:2208.09011* (2022).

[6] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. 2021. Sumcheck arguments and their applications. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*. Springer, 742–773.

[7] George EP Box and Mervin E Muller. 1958. A note on the generation of random normal deviates. *The annals of mathematical statistics* 29, 2 (1958), 610–611.

[8] Marta Cantero Gamito and Christopher T Marsden. 2024. Artificial intelligence co-regulation? The role of standards in the EU AI Act. *International journal of law and information technology* 32 (2024), eaae011.

[9] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. 2022. Membership inference attacks from first principles. In *2022 IEEE symposium on security and privacy (SP)*. IEEE, 1897–1914.

[10] Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P Ward. 2020. Reducing participation costs via incremental verification for ledger systems. *Cryptology ePrint Archive* (2020).

[11] Markus de Medeiros, Muhammad Naveed, Tancrède Lepoint, Temesghen Kahsai, Tristan Ravitch, Stefan Zetzsche, Anjali Joshi, Joseph Tassarotti, Aws Albarghouthi, and Jean-Baptiste Tristan. 2025. Verified foundations for differential privacy. *Proceedings of the ACM on Programming Languages* 9, PLDI (2025), 1094–1118.

[12] Damien Desfontaines. 2022. Tiny bits matter: precision-based attacks on differential privacy.

[13] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*. Springer, 265–284.

[14] Noel Elias. 2024. Lova: A Novel Framework for Verifying Mathematical Proofs with Incrementally Verifiable Computation. *Cryptology ePrint Archive* (2024).

[15] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.

[16] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. 2023. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1880–1894.

[17] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. 2015. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)* 62, 4 (2015), 1–64.

[18] Herumi. 2015. MCL Library. https://github.com/herumi/mcl/tree/0c31ab9648e81f54177325e55ea96dd8e9c8ba6b. Accessed: 2025-11-28.

[19] Matthew Jagielski, Jonathan Ullman, and Alina Oprea. 2020. Auditing differentially private machine learning: How private is private sgd? *Advances in Neural Information Processing Systems* 33 (2020), 22205–22216.

[20] Yufan Jiang, Maryam Zarezadeh, Tianxiang Dai, and Stefan Köpsell. 2025. AlphaFL: Secure Aggregation with Malicious 2 Security for Federated Learning against Dishonest Majority. *Proceedings on Privacy Enhancing Technologies* 4, 4 (2025), 348–368.

[21] Jiankai Jin, Eleanor McMurtry, Benjamin IP Rubinstein, and Olga Ohrimenko. 2022. Are we there yet? timing and floating-point attacks on differential privacy systems. In *2022 IEEE Symposium on security and privacy (SP)*. IEEE, 473–488.

[22] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and trends® in machine learning* 14, 1–2 (2021), 1–210.

[23] Amit Keinan, Moshe Shenfeld, and Katrina Ligett. 2025. How Well Can Differential Privacy Be Audited in One Run? *arXiv preprint arXiv:2503.07199* (2025).

[24] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report TR-2009. University of Toronto, Toronto, ON, Canada.

[25] Yuange Li and Xiong Fan. 2025. SUMMER: Recursive Zero-Knowledge Proofs for Scalable RNN Training. *Cryptology ePrint Archive* (2025).

[26] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.

[27] Johan Lokna, Anouk Paradis, Dimitar I Dimitrov, and Martin Vechev. 2023. Group and attack: Auditing differential privacy. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1905–1918.

[28] Fred Lu, Joseph Munoz, Maya Fuchs, Tyler LeBlond, Elliott Zaresky-Williams, Edward Raff, Francis Ferraro, and Brian Testa. 2022. A general framework for auditing differentially private machine learning. *Advances in Neural Information Processing Systems* 35 (2022), 4165–4176.

[29] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)* 39, 4 (1992), 859–868.

[30] Juan Ma, Hao Liu, Mingyue Zhang, and Zhiming Liu. 2024. VPFL: Enabling verifiability and privacy in federated learning with zero-knowledge proofs. *Knowledge-Based Systems* 299 (2024), 112115.

[31] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[32] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 691–706.

[33] Ilya Mironov. 2017. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*. IEEE, 263–275.

[34] Moni Naor, Omer Paneth, and Guy N Rothblum. 2019. Incrementally verifiable computation via incremental PCPs. In *Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part II 17*. Springer, 552–576.

[35] Arjun Narayan, Ariel Feldman, Antonis Papadimitriou, and Andreas Haeberlen. 2015. Verifiable differential privacy. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–14.

[36] Milad Nasr, Shuang Songi, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlin. 2021. Adversary instantiation: Lower bounds for differentially private machine learning. In *2021 IEEE Symposium on security and privacy (SP)*. IEEE, 866–882.

[37] Kaushik Nath and Palash Sarkar. 2018. Efficient Arithmetic In (Pseudo-)Mersenne Prime Order Fields. *Cryptology ePrint Archive, Paper 2018/985* (2018).

[38] Joseph P Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, et al. 2019. Duet: an expressive higher-order language and linear type system for statically enforcing differential privacy. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–30.

[39] Omer Paneth and Rafael Pass. 2022. Incrementally verifiable computation via rate-1 batch arguments. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1045–1056.

[40] Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. 2018. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.* 11, 1 (2018), 61–79.

[41] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. 2023. Elsa: Secure aggregation for federated learning with malicious actors. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1961–1979.

[42] César Sabater, Aurélien Bellet, and Jan Ramon. 2022. An accurate, scalable and verifiable protocol for federated differentially private averaging. *Machine Learning* 111, 11 (2022), 4249–4293.

[43] Ali Shahin Shamsabadi, Gefei Tan, Tudor Ioan Cebere, Aurélien Bellet, Hamed Haddadi, Nicolas Papernot, Xiao Wang, and Adrian Weller. 2024. Confidential-DPproof: Confidential Proof of Differentially Private Training. In *International Conference on Learning Representations (ICLR)*.

[44] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 3–18.

[45] Thomas Steinke, Milad Nasr, and Matthew Jagielski. 2023. Privacy auditing with one (1) training run. *Advances in Neural Information Processing Systems* 36 (2023), 49268–49280.

[46] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkllm: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 4405–4419.

[47] Paul Valiant. 2008. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*. Springer, 1–18.

[48] Yuxin Wang, Zeyu Ding, Daniel Kifer, and Danfeng Zhang. 2020. Checkdp: An automated and integrated approach for proving differential privacy or finding precise counterexamples. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 919–938.

[49] Jianqi Wei, Yuling Chen, Xiuzhang Yang, Yun Luo, and Xintao Pei. 2025. A verifiable scheme for differential privacy based on zero-knowledge proofs. *Journal of King Saud University Computer and Information Sciences* 37, 3 (2025), 1–15.

[50] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 733–764.

[51] Hengchu Zhang, Edo Roth, Andreas Haeberlen, Benjamin C Pierce, and Aaron Roth. 2019. Fuzzi: A three-level logic for differential privacy. *Proceedings of the ACM on Programming Languages* 3, ICFP (2019), 1–28.
[52] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. 2021. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 159–177.
[53] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. *Advances in neural information processing systems* 32 (2019).
[54] Wan Zhu. 2018. Classification of MNIST handwritten digit database using neural network. *Proceedings of the research school of computer science. Australian National University, Acton, ACT* 2601 (2018).

A Deferred Preliminaries

Definition 4 (Zero-Knowledge Arguments [16].)**.** *Let R be an NP relation with its corresponding language L_R. A zero-knowledge argument of knowledge for R is defined by a triple of algorithms (G, P, V), provided the following conditions are satisfied. We denote G as the setup algorithm, P as the prover, and V as the verifier.*

- ***Completeness:*** *For every pair $(x, w) \in R$ and public parameter $pp \leftarrow G(1^\lambda)$, we have $\Pr\left[\langle P(pp, w), V(pp)\rangle(x) = 1\right] = 1$.*
- ***Knowledge Soundness:*** *For any probabilistic polynomial-time (PPT) adversary A, there exists a PPT extractor ξ_A such that for all x and $pp \leftarrow G(1^\lambda)$:*

$$\Pr\left[w \leftarrow \xi_A(pp, x) : (x, w) \in R\right] \geq$$
$$\Pr\left[\langle A(pp), V(pp)\rangle(x) = 1\right] - negl(\lambda)$$

- ***Zero-Knowledge:*** *There exists a PPT simulator S_A such that for every adversary A, for all $(x, w) \in R$, and $pp \leftarrow G(1^\lambda)$:*

$$View\left(\langle P(pp, w), A(pp, z)\rangle(x)\right) \approx S_A(x, z)$$

An argument system is considered transparent *if the public parameter pp is a randomly generated string.*

Definition 5 (Polynomial Commitments)**.** *A polynomial commitment scheme (PCS) allows a prover $\mathcal{P}$ to commit to a polynomial and later reveal its evaluation at a specific input point. Formally, a PCS consists of the following procedures:*

- ***KeyGen:*** *Given the security parameter, the number of variables ℓ, and the degree bound d, this procedure generates and outputs the public parameters* pp*.*
- ***Commit:*** *On input a polynomial f, randomness r, and public parameters* pp*, this procedure outputs a commitment σ to the polynomial.*
- ***Open:*** *Given an evaluation point x, polynomial f, randomness r, and public parameters* pp*, this procedure computes the evaluation $y = f(x)$ along with an evaluation opening proof π.*
- ***Verify:*** *On input a commitment σ, evaluation point x, evaluation result y, and proof π, this procedure verifies the correctness of the evaluation and returns either accept or reject.*

A PCS is *hiding* if the commitment σ does not reveal any information about the polynomial f. It satisfies *evaluation binding* if no prover can open a commitment σ to two distinct evaluations at the same input point x. *Knowledge soundness* ensures that any prover generating a correct evaluation proof must possess knowledge of the polynomial underlying the commitment. Additionally, a *zero-knowledge* PCS guarantees that the evaluation proof π does not disclose any information about f beyond the evaluated value $f(x)$.

Definition 6 (Incrementally Verifiable Computation (IVC))**.** *An IVC protocol consists of three key algorithms:*

- ***Setup*** *(*G*): Given a security parameter, this algorithm outputs public parameters pp.*
- ***Prove*** *(*P*): Given an iteration counter i, initial input z_0, prior output z_{i-1}, auxiliary witness ω_{i-1}, proof π_{i-1}, and public parameters pp, the prover generates the next output z_i along with a proof π_i.*
- ***Verify*** *(*V*): Given an iteration counter i, initial input z_0, output z_i, proof π_i, and public parameters pp, this algorithm returns either accept or reject, depending on whether the proof is valid.*

Each proof π_i serves as a zero-knowledge argument for the following relation:

$$R_{IVC,i} = \Big\{ (i, z_0, z_i), (\omega_0, \ldots, \omega_{i-1}) \mid \forall k \in [i],\ z_{k+1} = F(z_k, \omega_k) \Big\}.$$

A widely adopted approach for constructing IVC schemes is to employ recursive composition of a succinct non-interactive proof system as a baseline. Let P_b and V_b denote the prover and verifier of this baseline proof system, respectively. Define an augmented function F_A as $(F(z_i, \omega_i), V_b(i, z_0, z_i, \pi_i)) \leftarrow F_A(i + 1, z_0, z_i, \omega_i, \pi_i)$. Here, the prover P is responsible for computing z_{i+1} from z_i and ω_i, while also invoking P_b to generate the proof π_{i+1} that verifies the correctness of the computation: $F_A(i + 1, z_0, z_i, \omega_i, \pi_i) = (z_{i+1}, 1)$. A key advantage of the IVC framework is its incremental proof design, which ensures that the computational overhead for the prover, proof size, and verification complexity remain independent of the total number of iterations [10, 14]. Instead, these parameters scale only with the computational complexity of the augmented function F_A, making IVC a highly scalable and efficient solution for verifying long-running computations.

The Sumcheck Protocol. We recall the classical sumcheck protocol of [29], which enables a verifier to check claims of the form $H = \sum_{b \in \{0,1\}^\ell} f(b)$ for a low-degree multivariate polynomial $f : \mathbb{F}^\ell \to \mathbb{F}$. The protocol proceeds over ℓ rounds, progressively reducing the multivariate summation claim to the evaluation of f at a single random point chosen by the verifier. For completeness, we present the standard interactive protocol in Fig. 6, which forms the core subroutine used within the GKR construction.

The GKR Protocol. The GKR protocol extends the sumcheck protocol to enable succinct verification of layered arithmetic circuits. Consider a layered arithmetic circuit C over a finite field $\mathbb{F}$ with depth d, where layer 0 represents the output layer and layer d corresponds to the input layer. Each gate in layer i takes two inputs from layer $i + 1$. We denote the number of gates in layer i as S_i, and define $s_i := \lceil \log S_i \rceil$. We consider the wiring predicates add_i and $mult_i$, defined as $add_i, mult_i : \{0,1\}^{s_i + 2s_{i+1}} \to \{0,1\}$, where $add_i(z, x, y) = 1$ if and only if gate z in layer i is an addition gate with inputs x and y from layer $i + 1$, and $mult_i(z, x, y) = 1$ if and only if gate z in layer i is a multiplication gate with inputs x and y from layer $i + 1$. Defining the gate value function $v_i : \{0,1\}^{s_i} \to \mathbb{F}$ such that $v_i(b)$ represents the output of the b-th gate in layer i, we

The protocol consists of ℓ rounds and proceeds as follows:

(1) In the initial round, the prover P transmits the polynomial:
$$f_1(x_1) := \sum_{b_2,\ldots,b_\ell \in \{0,1\}} f(x_1, b_2, b_3, \ldots, b_\ell).$$
The verifier V verifies whether $H = f_1(0) + f_1(1)$ and subsequently selects r_1 uniformly from the field $\mathbb{F}$.

(2) For each intermediate round i ($2 \le i \le \ell - 1$), the prover P transmits the polynomial:
$$f_i(x_i) := \sum_{b_{i+1},\ldots,b_\ell \in \{0,1\}} f(r_1, \ldots, r_{i-1}, x_i, b_{i+1}, \ldots, b_\ell).$$
The verifier V checks if $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$ and then selects r_i uniformly from $\mathbb{F}$.

(3) In the final round ($i = \ell$), the prover P sends the polynomial:
$$f_\ell(x_\ell) = f(r_1, r_2, \ldots, x_\ell).$$
The verifier V verifies whether $f_{\ell-1}(r_{\ell-1}) = f_\ell(0) + f_\ell(1)$. Next, the verifier selects the last challenge r_ℓ uniformly from $\mathbb{F}$. Given oracle access to f, the verifier accepts if and only if:
$$f_\ell(r_\ell) = f(r_1, r_2, \ldots, r_\ell).$$

Figure 6: Sumcheck protocol

derive the recurrence relation:

$$\tilde{v}_i(z) = \sum_{x,y\in\{0,1\}^{s_{i+1}}} \Big[\widetilde{add}_i(z,x,y)\cdot(\widetilde{v}_{i+1}(x)+\widetilde{v}_{i+1}(y)) + \widetilde{mult}_i(z,x,y)\cdot\widetilde{v}_{i+1}(x)\cdot\widetilde{v}_{i+1}(y)\Big] \quad (2)$$

The prover P can demonstrate the evaluation of the circuit C as follows. Assume that the verifier V has oracle access to $\widetilde{v}_0$ and $\widetilde{v}_d$, which represent the input and output, respectively. The verifier first selects a random challenge r_0 and computes $\widetilde{v}_0(r_0)$. Next, the prover and verifier execute the sumcheck protocol on Equation 2 for the case $i = 0$. At the conclusion of this round, the verifier requests two evaluations of $\widetilde{v}_1$ at specific points, say $r_{1,0}$ and $r_{1,1}$. Let the prover provide the values $y_0 = \widetilde{v}_1(r_{1,0})$ and $y_1 = \widetilde{v}_1(r_{1,1})$. The verifier then chooses new values for y_0 and y_1 from the finite field $\mathbb{F}$, and the parties proceed with another round of the sumcheck protocol to verify that the following equation holds:

$$\gamma_0\cdot y_0 + \gamma_1\cdot y_1 = \sum_{x,y\in\{0,1\}^{S^2}} (\gamma_0\cdot\widetilde{add}_1(r_{1,0},x,y) + \gamma_1\cdot\widetilde{add}_1(r_{1,1},x,y))$$
$$\cdot(\widetilde{v}_2(x)+\widetilde{v}_2(y)) + (\gamma_0\cdot\widetilde{mult}_1(r_{1,0},x,y) + \gamma_1\cdot\widetilde{mult}_1(r_{1,1},x,y))\cdot$$
$$\widetilde{v}_2(x)\cdot\widetilde{v}_2(y).$$

This simplifies the task to verifying two evaluations of $\widetilde{v}_2$. The protocol proceeds iteratively for d rounds, during which the verifier checks the final evaluations of $\widetilde{v}_d$ through its oracle access. The complete description of the protocol can be found in Fig. 7 in Appendix A. To ensure that the GKR protocol is zero-knowledge, zero-knowledge sumchecks [6, 50] and zero-knowledge polynomial commitments can be employed to facilitate oracle access to $\widetilde{v}_0$ and $\widetilde{v}_d$. Furthermore, by applying the Fiat-Shamir transform, the protocol [15] can be made non-interactive.

Let F be a finite field and let $C : F_{\text{in}} \to F_{\text{out}}$ represent a layered arithmetic circuit of depth d. The goal of Prover P is to convince Verifier V that the computation $C(\text{in})$ results in the output out, where in is the input provided by V and out is the corresponding output. We assume, without loss of generality, that both the input in and output out are padded to powers of 2. The protocol proceeds as follows:

(1) Verifier V selects a random element $z \in F$ and sends it to Prover P. Both parties compute $\widetilde{V}_0(z)$, where $\widetilde{V}_0$ denotes the multilinear extension of the output array out.

(2) Prover P and Verifier V execute a sumcheck protocol on the following equation:
$$\widetilde{V}_0(z) = \sum_{x,y\in\{0,1\}} \widetilde{add}_0(z,x,y).(\widetilde{V}_1(x)+\widetilde{V}_1(y)) + \widetilde{mult}_0(z,x,y).\widetilde{V}_1(x).\widetilde{V}_1(y)$$
At the conclusion of this protocol, Verifier V receives two evaluations, $\widetilde{V}_1(u^{(1)})$ and $\widetilde{V}_1(v^{(1)})$. Verifier V computes $\widetilde{add}_0(z,u^{(1)},v^{(1)})$ and $\widetilde{mult}_0(z,u^{(1)},v^{(1)})$, then verifies that the following equation holds:
$$\widetilde{add}_0(z,u^{(1)},v^{(1)})(\widetilde{V}_1(u^{(1)})+\widetilde{V}_1(v^{(1)})) + \widetilde{mult}_0(z,u^{(1)},v^{(1)}).\widetilde{V}_1(u^{(1)}).\widetilde{V}_1(v^{(1)}) = \text{last message of sumcheck.}$$

(3) For each round $i = 1, 2, \ldots, d$:
- Verifier V selects two random values $\alpha^{(i)}$ and $\beta^{(i)}$ and sends them to Prover P.
- Prover P and Verifier V execute a sumcheck protocol on the following equation:
$$\alpha^{(i)}\widetilde{V}_i(u^{(i)}) + \beta^{(i)}\widetilde{V}_i(v^{(i)}) = \sum_{x,y\in\{0,1\}} (\alpha^{(i)}\widetilde{add}_i(u^{(i)},x,y) + \beta^{(i)}\widetilde{add}_i(v^{(i)},x,y)).(\widetilde{V}_{i+1}(x)+\widetilde{V}_{i+1}(y))$$
$$+ (\alpha^{(i)}\widetilde{mult}_i(u^{(i)},x,y) + \beta^{(i)}\widetilde{mult}_i(v^{(i)},x,y)).\widetilde{V}_{i+1}(x).\widetilde{V}_{i+1}(y)$$
At the conclusion of this protocol, Verifier V receives two evaluations, $\widetilde{V}_{i+1}(u^{(i+1)})$ and $\widetilde{V}_{i+1}(v^{(i+1)})$. Verifier V then computes the following:
$$(\alpha^{(i)}\widetilde{add}_i(u^{(i)},u^{(i+1)},v^{(i+1)}) + \beta^{(i)}\widetilde{add}_i(v^{(i)},u^{(i+1)},v^{(i+1)})).$$
$$(\widetilde{V}_{i+1}(u^{(i+1)})+\widetilde{V}_{i+1}(v^{(i+1)})) + (\alpha^{(i)}\widetilde{mult}_i(u^{(i)},u^{(i+1)},v^{(i+1)}) + \beta^{(i)}\widetilde{mult}_i(v^{(i)},u^{(i+1)},v^{(i+1)})).\widetilde{V}_{i+1}(u^{(i+1)}).\widetilde{V}_{i+1}(v^{(i+1)})$$
If the computed result does not match the last message of the sumcheck, Verifier V returns 0.

(4) At the final layer d, Verifier V is provided with two claims, $\widetilde{V}_d(u^{(d)})$ and $\widetilde{V}_d(v^{(d)})$. Since Verifier V has access to the input in, it is able to compute $\widetilde{V}_d$ and evaluate it at the points $u^{(d)}$ and $v^{(d)}$. If the claims are valid, Verifier V returns 1; otherwise, it returns 0.

Figure 7: GKR protocol

B Proof of Theorem 4.2

Proof. **Completeness.** If the prover honestly executes $\mathcal{A}_{\text{DP-SGD}}$ with parameters $pp = (C, \eta, \sigma, b, T, L)$ on the committed dataset D, then each iteration satisfies:

(1) seed correctness $s = k \oplus H(\text{Com}(k))$,
(2) per-example gradient correctness $g_i = \nabla_\theta L(\theta_{t-1}, (x_i, y_i))$,
(3) clipping $\hat{g}_i = g_i / \max(1, \|g_i\|_2 / C)$,
(4) averaging $\tilde{\hat{g}}_t = \frac{1}{b}\sum_{i\in B_t}\hat{g}_i$,
(5) noise generation $n_t \sim \mathcal{N}(0, \sigma^2 C^2 I)$ via the Box–Muller circuit,

(6) update equation $\theta_t = \theta_{t-1} - \eta(\tilde{\hat{g}}_t + n_t)$.

All these constraints are enforced by the zero-knowledge proof system. By completeness of the underlying argument system, an honest prover is always accepted.

Knowledge Soundness. Suppose a PPT prover convinces the verifier in Π_{VERIDP}. By knowledge soundness of sumcheck/GKR and polynomial commitments, there exists an extractor that recovers valid witnesses consistent with all constraints: namely the per-example gradients $\{g_i\}$, clipped versions $\{\hat{g}_i\}$, and noise vectors $\{n_t\}$ generated from the committed seed. Thus, any accepting proof yields $w = (\{g_t\}, \{n_t\})$ such that $(x, w) \in R_{\text{DP-SGD}}$, up to negligible soundness error.

Privacy Enforcement. The seed s is deterministically derived from the dataset commitment, and the Box-Muller circuit guarantees that $n_t \sim \mathcal{N}(0, \sigma^2 C^2 I)$ at each iteration. Hence, any accepting transcript certifies that every update uses clipped gradients and Gaussian noise with the declared variance, ensuring the (ϵ, δ)-DP guarantee required by Definition 2.

Zero-Knowledge. The protocol employs zero-knowledge sumcheck/GKR and zero-knowledge polynomial commitments, which hide the dataset, gradients, randomness, and noise. By the ZK property of these primitives, there exists a simulator that, given only the public input $x = (\text{Com}(D), pp, \theta_0, \theta_T)$, produces transcripts indistinguishable from real executions. Thus the verifier learns nothing beyond correctness of DP-SGD execution.

So, Π_{VERIDP} satisfies completeness, knowledge soundness, zero-knowledge, and privacy enforcement, and therefore constitutes a Zero-Knowledge Proof of DP-SGD for $R_{\text{DP-SGD}}$ as in Definition 2. □