FALAFEL: Modular Zero-Knowledge Proofs of Training in the Federated Setting

Tariq Bontekoe*, Sven Bootsma†, Vincent Dunning†‡, Thom Sijpesteijn†, Thomas Attema†§
* *University of Groningen, Bernoulli Institute, Groningen, The Netherlands*
† *Netherlands Org. for Applied Scientific Research (TNO), The Hague, The Netherlands*
‡ *University of Twente, Enschede, The Netherlands*
§ *CWI, Cryptology Group, Amsterdam, The Netherlands*
t.h.bontekoe@rug.nl {sven.bootsma, vincent.dunning, thom.sijpesteijn, thomas.attema}@tno.nl

***Abstract*—We introduce FALAFEL, a modular scheme for *Fast, Authenticated, Locally Attested FEderated Learning*, with which parties can create a zero-knowledge proof of training (zkPoT) for Federated Learning (FL). The proof guarantees active security during the federated training process as well as publicly verifiable correctness of the final, trained model. All without introducing any additional leakage about the local datasets or intermediate local model states beyond what is inherently leaked by the federated learning process (e.g., through shared model updates).**

Specifically, our approach targets FL of (deep) neural networks with a centralized server for weight updates. Our zkPoT not only offers attestation for local training steps, but also for the centralized weight update, as well as taking into account input authenticity by introducing a trusted auditor. This way an external verifier can check the entire training process, from dataset to final model.

In contrast to prior work on zkPoTs, our construction solely relies on well-understood cryptographic assumptions and primitives, is highly parallelizable, and takes a modular approach. This modular (commit-and-prove) approach uses several novel core proof components, which could be swapped for other building blocks if desired. We show that, for LeNet, we generate a zkPoT of 70 kB in roughly 150 seconds for a single training round. FALAFEL's prover time is in line with prior work, and its proof size is significantly smaller (10–15×), without relying on less-understood assumptions or instantiating Fiat–Shamir using arithmetic hash functions.

1. Introduction

Artificial intelligence and machine learning (ML) models have demonstrated significant added value in prediction, classification, and generation tasks. The ever-increasing attention to ML, particularly (deep) neural networks ((D)NNs), has led to the large-scale integration of such systems into real-world applications. However, this integration has also highlighted challenges and raised concerns regarding data authenticity and computational integrity [1]. The data hunger of these models has additionally raised privacy concerns, particularly when they are trained on sensitive data, such as in healthcare or finance. In many such sensitive applications, datasets are also distributed across multiple entities, where data sharing is not always possible due to legal and privacy constraints.

Federated learning (FL) is proposed as an efficient, privacy-friendly solution that enables each entity to train a local model on its own dataset and periodically share model parameters with a trusted server for aggregation into a global model, without local data ever leaving the premises. Whilst FL is more privacy-friendly, it also raises additional concerns regarding authenticity and integrity. E.g., malicious parties may attempt to infer sensitive information about other parties' data or try to manipulate the final model. Such manipulation can occur by tampering with local training data, deviating from the prescribed training procedure, or, if the server is untrusted, by incorrectly performing the aggregation step.

Recent work shows that traditional verification approaches fall short for ML systems, as they cannot simultaneously provide strong guarantees on training correctness while preserving the privacy of models and datasets, especially in federated settings where data and computation are distributed [2]. To address these issues and provide both active security and public verifiability of the federated learning training process, we introduce FALAFEL, a zero-knowledge proof of training (zk-PoT) [3] for federated learning with input authenticity, which reveals no additional information about clients' local datasets or model weights to other parties. FALAFEL consists of several commit-and-prove zero-knowledge proofs (ZKPs) [4] that are either novel or adapted from existing work. Our proofs are based on standard, well-understood cryptographic assumptions: discrete-log and random oracle model; in particular, we do not rely on trusted setups or knowledge-of-exponent assumptions, and instantiate Fiat–Shamir using conventional hash functions rather than less-vetted *zk-friendly* alternatives. Our approach is modular and parallelizable and, above all, provides strong privacy and security guarantees while achieving practical efficiency.

1.1. Related Work

We focus on discussing existing solutions for *training verification* and *verification in federated settings*. Additional related work can be found in Appendix A.

Training Verification. In principle, zk-PoT schemes can be constructed using generic ZKP systems [5], [6], [7], but

their performance for practical neural networks is prohibitive. In particular, most circuit-based succinct zero-knowledge proofs (zk-SNARKs) incur very high prover costs, both in terms of proof-generation time and memory usage. Contrastingly, constructions that achieve more efficient provers typically do so at the cost of impractically large proof sizes and verification times. To address these limitations, several specialized schemes have been proposed to verify a (D)NN's training process. *zkDL* [8] introduces *zkReLU* for exact ReLU verification using sumcheck proofs and *FAC4DNN* for modeling training as arithmetic circuits. *Kaizen* [3] presents a scheme for proving correctness of training on committed models and datasets using GKR-style sumcheck proofs with recursive composition. Garg et al. [9] combine zk-SNARKs with MPC-in-the-head to obtain a zkPoT for logistic regression; however, there is no clear path toward an efficient extension of their approach to neural networks. *ZkAudit* [10] proposes a protocol for trustless audits of DNNs, including zkPoT capabilities, while preserving the privacy of model weights and training data. Their solution, however, is not applicable to our use case, and extending it accordingly would lead to impractical efficiency.

Ergo, the works most suitable to our setting are along the lines of *zkDL* and *Kaizen*, both of which rely on sumcheck protocols and/or GKR-style proofs to achieve practical efficiency. However, unlike our approach, *zkDL* is monolithic, i.e., there is no possibility to take advantage of more efficient building blocks for proving specific sub-relations, and less efficient than *Kaizen*. Consequently, *Kaizen* appears to be the most practically efficient solution applicable to our setting, but only does so by relying on non-standard cryptographic assumptions, in particular the use of arithmetic hash functions to instantiate the random oracle in the Fiat–Shamir model. As a benefit over our work, as discussed in Section 6, *Kaizen* has a substantially more efficient verification process and may therefore be preferable for very large datasets or models. However, in federated settings such large datasets are uncommon, making our approach more suitable for FL. Moreover, our scheme's proof size is significantly smaller (10–15×) and its prover time is in line with *Kaizen*, whilst relying solely on well-understood assumptions and primitives.

Verification in Federated Learning. Several schemes address verification in federated settings. *VPNNT* [11] enables clients to prove that their model updates are derived from the global model without revealing private data, using sumcheck proofs. However, their setting assumes a trusted server, which is undesirable in most real-world scenarios. Moreover, we were unable to reproduce their results, as no public implementation is available. Alternatively, *Federify* [12] proposes a blockchain-based federated learning framework that uses zk-SNARKs for Gaussian Bayes models, where the blockchain serves as a coordination and audit layer for model updates and proofs. This approach relies on generic zk-SNARKs for training verification, and we do not see a clear path to scaling it to more complex models.

1.2. Contributions

We introduce FALAFEL, a novel protocol for verifiable federated learning scenarios where training occurs across multiple distributed parties. The protocol secures the complete pipeline from dataset to joined trained model and offers several key contributions compared to prior work:

- **Input authenticity.** We incorporate the possibility for external auditors to audit and sign datasets before training to ensure training data integrity and provenance.
- **Randomized batching.** To the best of our knowledge, we are the first to introduce an efficient proof for generating locally randomized mini-batches from a dataset, as used, for example, in mini-batch gradient descent.
- **Modular design.** Our protocol is highly modular and consists of interchangeable building blocks, enabling flexible instantiation with potential novel cryptographic primitives.
- **Scaling proofs.** We introduce an adaptation of aggregated Bulletproofs-style range proofs [13] to prove correctness of fixed-point scaling, which may be of independent interest.
- **Security model.** Compared to prior work, we rely solely on well-understood cryptographic assumptions. Specifically, we instantiate the Fiat–Shamir heuristic using well-vetted hash functions and rely only on discrete-logarithm-type assumptions. Moreover, compared to the closest related work [3], we eliminate recursive proof composition, thereby removing non-standard security assumptions and enabling a clean and well-understood security analysis, while also reducing prover complexity [14]. Furthermore, and independently of the above, we prove security of our core components in a more rigorous manner (using modern techniques) than prior work.
- **Parallelizability & balanced prover-verifier complexity.** Our overall design, as well as its building blocks, is highly parallelizable, leading to good practical efficiency. In addition, we deliberately avoid minimizing verifier costs at the expense of a disproportionate increase in prover complexity, since in a federated learning setting each party acts as both prover and verifier. Instead, we aim to strike a balanced trade-off between prover and verifier complexity.
- **Performance.** We fully implement and publicly release FALAFEL. Through extensive benchmarking, we show that our solution scales well and achieves practical performance. Concretely, we show that FALAFEL generates a zkPoT for a LeNet training round in $\sim$150 seconds, which is competitive with the state-of-the-art, and without relying on heavy amortization. Furthermore, FALAFEL achieves a significantly smaller proof size of $\sim$70 kB.

1.3. Organization

We introduce notation and relevant background in Section 2, and provide a high-level overview of FALAFEL and its threat model in Section 3. Our core proof components are presented in Section 4 and combined into the main protocol in Section 5. Formal security proofs are deferred to the appendix. In Section 6, we evaluate the performance of FALAFEL, and we conclude in Section 7.

2. Preliminaries

Let $\mathbb{G}$ be a multiplicative cyclic group of prime order p. We denote by $\mathbb{Z}_p$ the additive group of integers modulo p; when equipped with multiplication modulo p, this group forms the finite field $\mathbb{F}_p$. When the prime is clear from context, we simply write $\mathbb{F}$. We view $\mathbb{G}$ as a one-dimensional vector space over $\mathbb{F}$, where the group operation corresponds to vector addition (i.e., $gh \in \mathbb{G}$ for $g, h \in \mathbb{G}$) and scalar multiplication is written as exponentiation, i.e., g^x for $g \in \mathbb{G}$ and $x \in \mathbb{F}$. We follow the convention that $0 \in \mathbb{N}$.

Vectors and matrices. Vectors are written in bold lowercase letters (e.g., $\boldsymbol{x}$), and matrices in bold uppercase letters (e.g., $\boldsymbol{X}$). Given a matrix $\boldsymbol{X}$ with n rows and m columns, we write x_{ij} for the element in row $i \in [n]$ and column $j \in [m]$, where $[n] \coloneqq \{1, \ldots, n\}$. We denote by $\boldsymbol{x}_i$ the i-th row or column of $\boldsymbol{X}$ depending on context. Matrix multiplication is denoted as $\boldsymbol{A} \cdot \boldsymbol{B}$ and the Hadamard (or element-wise) product as $\boldsymbol{A} \circ \boldsymbol{B}$. For $k \in \mathbb{Z}_p$ and $n \in \mathbb{N}$, we write $\boldsymbol{k}^n \coloneqq (1, k, \ldots, k^{n-1})$ and $\boldsymbol{k}^{-n} \coloneqq (1, k^{-1}, \ldots, k^{-n+1})$. For $a \in \mathbb{N}$ and $n \in \mathbb{N}$, we write $a^{\times n}$ to denote the constant vector $(a, \ldots, a) \in \mathbb{N}^n$. For a vector $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\boldsymbol{a} \in \mathbb{Z}^n$, we define $\boldsymbol{g}^{\boldsymbol{a}} \coloneqq \prod_{i=1}^{n} g_i^{a_i}$; analogously for $b \in \mathbb{Z}$, we write $\boldsymbol{g}^b \coloneqq \prod_{i=1}^{n} g_i^b$.

2.1. Neural-Network Architecture

We assume familiarity with the basics of neural networks. We focus on fully-connected feed-forward neural networks with L layers, where each layer $\ell \in [L]$ contains n_ℓ neurons. Layers $\ell = 1, \ldots, L-1$ are hidden layers and $\ell = L$ is the output layer. An input vector $\boldsymbol{x} \in \mathbb{R}^d$ has input dimension d. Rather than separating weights and biases, we combine them into augmented matrices $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{n_\ell \times (n_{\ell-1}+1)}$ for $\ell \in [L]$, with $n_0 \coloneqq d$. The first column stores the bias values and each remaining column stores the weights connecting one neuron from the previous layer to all neurons in the current layer. We denote all trainable network parameters by $\boldsymbol{\mathcal{W}} = \left(\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)}\right)$.

For our protocol and its implementation we will use mini-batch gradient descent with a fixed learning rate for the weight updates, and the mean squared error (MSE) loss. Also, our choice of activation function $\mathsf{act}(\cdot)$ will be the ReLU function given by $\mathrm{ReLU}(x) = \max(0, x)$. Possible extensions are discussed in Section 7.

Dataset & Training Matrices. A dataset $D = (\boldsymbol{X}, \boldsymbol{Y})$ is represented by $\boldsymbol{X} = [\,\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_{\text{samples}}}\,] \in \mathbb{R}^{n_0 \times N_{\text{samples}}}$, and $\boldsymbol{Y} = [\,\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N_{\text{samples}}}\,] \in \mathbb{R}^{n_L \times N_{\text{samples}}}$.

We use mini-batch training with batch size S_{batch} for E epochs. For simplicity we assume that $N_{\text{batch}} \coloneqq \frac{N_{\text{samples}}}{S_{\text{batch}}}$ is an integer. The learning rate is denoted by η. The i-th training batch is given by $B_i = (\boldsymbol{X}_i, \boldsymbol{Y}_i)$, with $\boldsymbol{X}_i$ and $\boldsymbol{Y}_i$ consisting of the columns of $\boldsymbol{X}$ and $\boldsymbol{Y}$ indexed by $\mathcal{I}_i \in \mathbb{N}^{S_{\text{batch}}}$, where $\{\mathcal{I}_i\}$ is a valid partition of $[N_{\text{samples}}]$.

Below, we explain how we define the different parts of the local training process (forward pass, backward pass, weight update) for the (D)NNs we consider. Specifically, during training step i and for layer $\ell \in [L]$ we distinguish the following matrices:

- $\boldsymbol{T}_i^{(\ell)}$: pre-activation values, i.e., input to activation function;
- $\boldsymbol{U}_i^{(\ell)}$: post-activation values;
- $\boldsymbol{W}_{i-1}^{(\ell)}$: weight matrix (incl. bias) before step i;
- $\boldsymbol{R}_i^{(\ell)}$: backward chain term, collecting all factors arising from the product rule in backpropagation;
- $\boldsymbol{T}_i'^{(\ell)}$: derivative of the activation function, i.e., for ReLU we have $\boldsymbol{T}_i'^{(\ell)} = \mathbf{1}_{x \geq 0}(\boldsymbol{T}_i^{\ell})$, where $\mathbf{1}_{x \geq 0}$ is 1 for non-negative input and 0 otherwise;
- $\overline{\boldsymbol{G}}_i^{(\ell)}$: average (over the mini-batch) weight gradient.

Forward pass. In the i-th training step, let $\boldsymbol{U}^{(0)} \coloneqq \boldsymbol{X}_i$. During the forward pass, for each layer $\ell \in [L]$ we first compute $\boldsymbol{T}_i^{(\ell)} = \boldsymbol{W}_i^{(\ell)} \begin{bmatrix} \mathbf{1}^\top \\ \boldsymbol{U}_i^{(\ell-1)} \end{bmatrix}$ and then $\boldsymbol{U}_i^{(\ell)} = \mathsf{act}^{(\ell)}\left(\boldsymbol{T}_i^{(\ell)}\right)$. Using these unified weight matrices minimizes the number of matrix operations in the training phase, to compute the exact same thing, thereby reducing the cost of our solution.

Backward pass. In the i-th training step, let $\boldsymbol{R}_i^{(\ell+1)} \coloneqq \frac{2}{S_{\text{batch}}}(\boldsymbol{U}_i^{(\ell)} - \boldsymbol{Y}_i)$. During the backward pass, for each layer $\ell \in [L]$ we first compute $\overline{\boldsymbol{G}}_i^{(\ell)} = (\boldsymbol{R}_i^{\ell+1} \circ \boldsymbol{T}_i'^{(\ell)}) \cdot (\boldsymbol{U}_i^{(\ell-1)})^\top$, and then $\boldsymbol{R}_i^{(\ell)} = (\boldsymbol{W}_{i-1}^{(\ell)})^\top \cdot (\boldsymbol{R}_i^{(\ell+1)} \circ \boldsymbol{T}_i'^{(\ell)})$.

Weight update. At the end of each training step the weights are updated according to: $\boldsymbol{W}_i^{(\ell)} = \boldsymbol{W}_{i-1}^{(\ell)} - \eta\, \overline{\boldsymbol{G}}_i^{(\ell)}$.

Federated learning. FL [15], [16] is a collaborative approach for training ML models, enabling a set of parties — each with their own, private dataset — to train a global model without revealing their respective datasets. In round r of the protocol, the server S distributes the current global model to all parties. Each client P_k then locally updates this global model, by training it on its local dataset D_k and only shares the resulting local weights $\boldsymbol{\mathcal{W}}_{k,r}$ with S. Then, S aggregates all received updates to improve the global model. In our case, we take the average of all $\boldsymbol{\mathcal{W}}_{k,r}$. The new global weights $\boldsymbol{\mathcal{W}}_{\text{glob},r}$ are then shared with each P_k, who uses their local dataset to update it once more in round $r+1$, in this way defining an iterative process.

We observe that FL has its limitations regarding the provided level of privacy [17]. These limitations can be mitigated using other privacy-enhancing technologies, though such approaches are beyond the scope of the current paper; we take the privacy guarantees of FL as is.

2.2. Commitments, Proofs, and Signatures

Commitments. A commitment scheme $\mathsf{Com}(m, r)$ takes as input a value m and randomness r, and outputs a commitment cm. The pair (m, r) is called the opening of a commitment. A secure commitment scheme should satisfy:

- *Binding*: Given $\mathsf{cm} \coloneqq \mathsf{Com}(m, r)$ it is hard to find $(m'r')$, with $m' \neq m$, such that $\mathsf{Com}(m', r') = \mathsf{cm}$;
- *Hiding*: For any $x_0 \neq x_1$, and given $\mathsf{cm}_b \coloneqq \mathsf{Com}(x_b, r)$, it is hard to determine b (with probability more than 0.5).

In this work, we consider the Pedersen (vector) commitment scheme [18]. Given $\mathbb{G}$ of prime order p and generators $g_1, \ldots, g_n, h \in \mathbb{G}$, one commits to a message (vector) $\boldsymbol{m} \in \mathbb{F}^n$, by sampling a random element $r \in \mathbb{F}$ and computing $\mathsf{Com}(\boldsymbol{m}, r) \coloneqq h^r \prod_i g_i^{m_i}$. This scheme is perfectly hiding and computationally binding under the discrete log problem, and is additively homomorphic, i.e., $\mathsf{Com}(m_1, r) + \mathsf{Com}(m_2, r) = \mathsf{Com}(m_1 + m_2, r)$.

Zero-knowledge Proofs. Our constructions rely heavily on zero-knowledge proofs of knowledge; for brevity, we omit *of knowledge*. These allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that it knows a secret witness w for a given, public statement x, such that the pair satisfies some NP-relation $\mathcal{R}$, i.e., $(x; w) \in \mathcal{R}$. A ZKP should satisfy the following (informal) definitions: (1) *Completeness*: an honest prover can convince an honest verifier of the validity of a true statement; (2) *(Knowledge) soundness*: a dishonest prover cannot trick an honest verifier into accepting a false statement (i.e., there exists an extractor that, given access to the prover's state, can produce a valid witness); and (3) *Zero-knowledge*: the verifier gains no additional knowledge beyond the validity of the statement itself. We list formal definitions in Appendix B.

Multilinear Extensions. An extension of a function $f\colon \{0,1\}^\ell \to \mathbb{F}$ is an ℓ-variate polynomial g over $\mathbb{F}$, with $g(\boldsymbol{x}) = f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{0,1\}^\ell$. Any such f has a *unique* multi-linear extension (MLE) $\widetilde{f}$.

Sumcheck Protocol & Hyrax. We will often view a vector $\boldsymbol{x} \in \mathbb{F}^n$ as a function $\boldsymbol{x}\colon \{0,1\}^{\log(n)} \to \mathbb{F}$ mapping indices to vector entries, and denote its multilinear extension (MLE) by $\widetilde{\boldsymbol{x}}\colon \mathbb{F}^{\log(n)} \to \mathbb{F}$. Since we primarily work with matrices, we first map a matrix $\boldsymbol{X}$ to a vector $\boldsymbol{x}$, and view that as a function. The MLE $\widetilde{\boldsymbol{X}}$ of this function will be the MLE of $\boldsymbol{X}$. For clarity, we assume throughout that $n = 2^\ell$, which in practice can be achieved by zero-padding.

We are often interested in proving correctness of $H = \sum_{\boldsymbol{x} \in \{0,1\}^\ell} f(\boldsymbol{x})$: the sum of a function over the boolean hypercube. This is efficiently done using the (non-zk) sumcheck protocol [19]. Given an ℓ-variate polynomial f over $\mathbb{F}$, the sumcheck protocol allows $\mathcal{P}$ to convince $\mathcal{V}$, by reducing the claim about H to one about the value of $\widetilde{f}(\boldsymbol{r})$, for random $\boldsymbol{r} \in \mathbb{F}^\ell$ chosen by $\mathcal{V}$. This requires ℓ rounds and $\mathcal{V}$'s runtime is $\mathcal{O}(\sum_{i=1}^{\ell} \deg_i(f))$ plus the evaluation cost of $\widetilde{f}(\boldsymbol{r})$.

Hyrax [20] introduces mechanisms to efficiently make sumcheck protocols zero-knowledge while relying only on standard cryptographic assumptions. They achieve this using an efficient ZKP to execute the ℓ sumcheck rounds, and by transforming the most expensive step—evaluating $\widetilde{f}(\boldsymbol{r})$—using a *square-root commitment scheme* (see Section 4.2).

Digital Signatures. A signature scheme consists of three algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. KeyGen generates a public verification key pk and a secret signing key sk. The signing algorithm Sign uses sk, takes a message m and outputs a signature σ, which can be verified using Verify and pk. We require the scheme to be secure against *existential forgeries under a chosen-message attack* (EUF-CMA), meaning that it is computationally hard to produce a valid pair (m', σ') for a *new* message m'. We instantiate this primitive using Schnorr signatures [21], which are efficient, compatible with our proof techniques, and rely on well-understood assumptions.

3. Overview

We present a high-level overview of FALAFEL. FALAFEL enables actively secure, federated training of a global model on auditor-authenticated client datasets, and allows any external party to verify correctness of training without revealing private data or intermediate model states.

3.1. System Model

The FALAFEL protocol, illustrated in Figure 1, consists of three phases: (1) offline setup phase, (2) training phase, and (3) usage phase. Below, we give a high-level overview. Our novel and adapted core components for the proof of the training phase will be discussed in Section 4 and a full specification of FALAFEL is presented in Section 5.

Offline Phase. The clients and server agree on the NN architecture and public parameters pp, which remain fixed for the remainder of the protocol. Secondly, an auditor verifies the correctness and authenticity of client datasets. For an approved dataset D_k, it signs the corresponding commitment cm_{D_k}, yielding $\sigma_{\mathsf{cm}_{D_k}}$.

Training phase. This phase starts with each client P_k broadcasting their signed commitments $(\mathsf{cm}_{D_k}, \sigma_{\mathsf{cm}_{D_k}})$, which are verified by all parties. In round r, each P_k trains on their private dataset D_k, starting from the current global model weights $\boldsymbol{\mathcal{W}}_{\mathrm{glob},r}$. During this procedure, they also produce a proof $\pi_{k,r}$ attesting that the training was performed correctly.

The updated local weights $\boldsymbol{\mathcal{W}}_{k,r}$ are sent to the server, while the proofs are broadcast. All clients and the server verify all received proofs before proceeding to the next round.

The server aggregates the updated local weights into new global weights and produces a publicly verifiable proof $\pi_{\mathrm{glob},r}$ attesting to correct aggregation. The server distributes the updated global model to each client, while the aggregation proofs are broadcast. Each client verifies all proofs before continuing. This process repeats until training terminates, yielding a final trained model.

Usage phase. The final trained model is published together with a public proof transcript. Any external verifier can use the public parameters and this transcript to verify that the global model was obtained by correctly executing federated training on auditor-authenticated datasets.

3.2. Threat Model

Next, we describe the threat model for FALAFEL by discussing each of the actors. We assume that all parties have access to an authenticated communication channel.

Auditor. The auditor is an honest, non-colluding party. The auditor's verification keys are publicly known and distributed through a trusted public-key infrastructure (PKI).

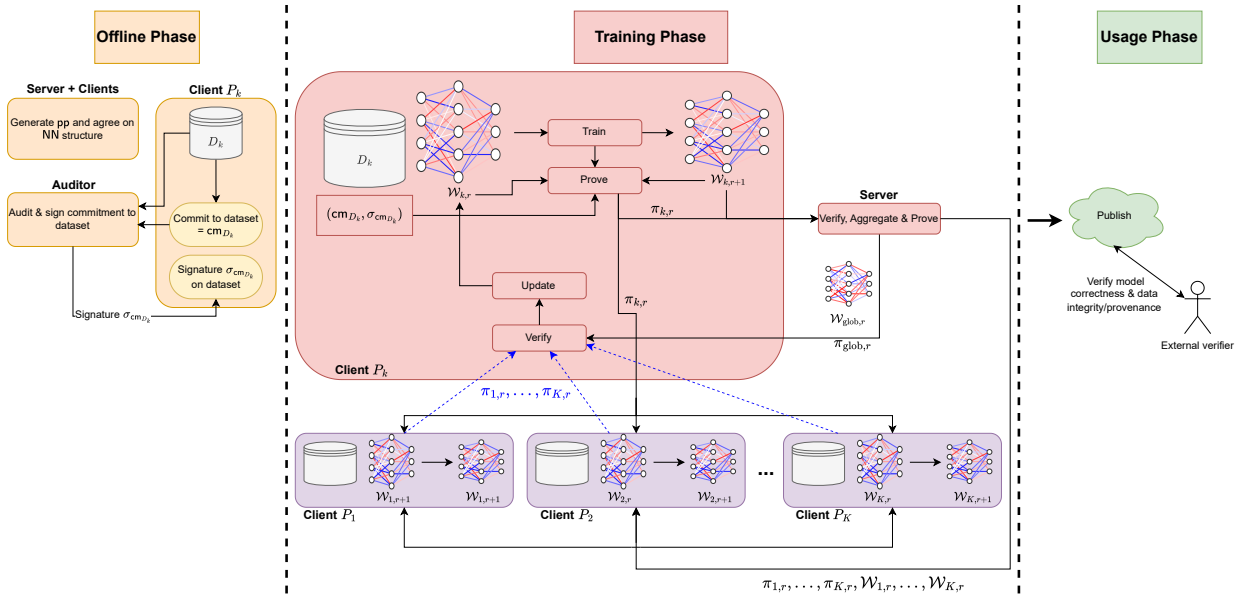

Figure 1. **FALAFEL protocol overview.** A depiction of the *offline phase*, round r of the *training phase*, and the *usage phase*.

Clients. Each client's dataset must remain private from everyone but the auditor. Up to and including all K clients may act maliciously by: (1) using different or unauthenticated data during training; (2) deviating from the prescribed protocol; (3) colluding with other clients to compromise model integrity or extract private information. Since FALAFEL is publicly verifiable, any such behavior will lead to an invalid proof.

Server. Most FL protocols assume an honest-but-curious server. In contrast, we consider a stronger threat model, where the server may act maliciously by deviating from the prescribed aggregation algorithm. We do not consider leakage or collusion attacks, i.e., we assume the server does not share received local updates with clients or external entities.

External verifiers. Any party with access to the training transcript can verify that the global model was correctly computed from auditor-authenticated data. Verifiers should learn nothing about individual client datasets or local/global model states beyond what is revealed by the final model.

4. Core Proof Components

In this section, we present the core proof components that are used to construct the FALAFEL protocol described in Section 5. Some of these components rely on (adaptions from) prior work, most notably Hyrax [20] and Bulletproofs [13], while others are introduced specifically for FALAFEL. Wherever a component is an adaptation of existing work, we first reiterate the relevant details, highlight the precise points where modifications are necessary, and detail its usage in our setting. Also, we prove security (Appendix E) using modern techniques, enabling clean proofs for composed protocols.

Several protocols make use of a $\mathsf{proof}_{\log}$-of-dot-product on $(\mathsf{cm}_{\boldsymbol{x}}, \boldsymbol{a}, \mathsf{cm}_y)$, as defined in [20, A.3]. This adaptation of Bulletproofs is used to prove that $y = \langle \boldsymbol{x}, \boldsymbol{a} \rangle$, given public $\boldsymbol{a}$, a commitment $\mathsf{cm}_{\boldsymbol{x}}$, and a commitment cm_y.

4.1. Shuffle & Split

Before diving into the actual training process, we present a novel interactive ZKP to verify that a dataset has been shuffled and split into mini-batches correctly, tailored to the square-root commitment scheme we will use (see Section 4.2). The proof ensures the mini-batches together contain exactly the same data points as the original dataset — without duplicating or omitting any entries — and that each batch has size S_{batch}, with zero-padding if necessary.

Using the notation of Section 2, we consider a set of mini-batches $\{(\boldsymbol{X}_i, \boldsymbol{Y}_i)\}_{i \in [N_{\text{batch}}]}$, where all $\boldsymbol{X}_i$ and $\boldsymbol{Y}_i$ are matrices with S_{batch} columns. Let $\boldsymbol{X}'$ and $\boldsymbol{Y}'$ denote the concatenation of the batch matrices $\boldsymbol{X}_i$ and $\boldsymbol{Y}_i$, respectively, and let ψ be a public permutation of the index set $[N_{\text{samples}}]$.

For given commitments, $\mathcal{P}$ needs to prove knowledge of a dataset $(\boldsymbol{X}, \boldsymbol{Y})$ and such mini-batches, and that the batched data is consistent with the dataset, in the sense that the j-th column of $\boldsymbol{X}'$ equals the $\psi(j)$-th column of $\boldsymbol{X}$. That is, $\boldsymbol{X}'$ is a permutation of $\boldsymbol{X}$ that has been split into mini-batches of the prescribed size. These requirements are captured in Theorem 4.1.

Definition 4.1 (Shuffle & Split Relation)**.** *The shuffle & split relation $\mathcal{R}_{\mathsf{shf\text{-}spl}}$ is defined over public inputs x and witnesses w. We have $(x; w) \in \mathcal{R}_{\mathsf{shf\text{-}spl}}$ if and only if*

$$x = \left(\mathsf{cm}_{\boldsymbol{X}}, \mathsf{cm}_{\boldsymbol{Y}}, \{\mathsf{cm}_{\boldsymbol{X}_i}, \mathsf{cm}_{\boldsymbol{Y}_i}\}_{i \in [N_{\text{batch}}]}, \psi\right),$$
$$w = \left(\boldsymbol{X}, r_{\boldsymbol{X}}, \boldsymbol{Y}, r_{\boldsymbol{Y}}, \{r_{\boldsymbol{X}_i}, r_{\boldsymbol{Y}_i}\}_{i \in [N_{\text{batch}}]}\right),$$

such that

$$\mathsf{cm}_{\boldsymbol{X}} = \mathsf{Com}(\mathsf{Vec}(\boldsymbol{X}), r_{\boldsymbol{X}}) \text{ and } \mathsf{cm}_{\boldsymbol{Y}} = \mathsf{Com}(\mathsf{Vec}(\boldsymbol{Y}), r_{\boldsymbol{Y}}),$$

where $\mathsf{Vec}(\cdot)$ *creates a column-major order vectorization and for all* $i \in [N_{\text{batch}}]$

$$\mathsf{cm}_{\boldsymbol{X}_i} = \mathsf{Com}((\boldsymbol{x}_{\psi((i-1)\cdot S_{\text{batch}}+1)} \| \dots \| \boldsymbol{x}_{\psi(i\cdot S_{\text{batch}})}), r_{\boldsymbol{X}_i}),$$
$$\mathsf{cm}_{\boldsymbol{Y}_i} = \mathsf{Com}((\boldsymbol{y}_{\psi((i-1)\cdot S_{\text{batch}}+1)} \| \dots \| \boldsymbol{y}_{\psi(i\cdot S_{\text{batch}})}), r_{\boldsymbol{Y}_i}).$$

Algorithm 1 defines the interactive ZKP for $\mathcal{R}_{\mathsf{shf\text{-}spl}}$. Given a verifier challenge $z \in \mathbb{F}^*$, $\mathcal{P}$ encodes the shuffled dataset into two scalar values t_1 and t_2 by taking random linear combinations of the permuted columns of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively, and commits to both values. Next, the commitments to all mini-batches are aggregated into a single commitment $\mathsf{cm}_{\hat{D}}$ using the same powers of z. A dot-product proof establishes consistency between $\mathsf{cm}_{\hat{D}}$ and $\mathsf{cm}_{t_1} \cdot \mathsf{cm}_{t_2}$, ensuring that the mini-batches jointly encode the same shuffled dataset. Two additional dot-product proofs verify consistency between the commitments to the original dataset, combined according to the public permutation ψ, and the commitments cm_{t_1} and cm_{t_2}.

The exponent $S_{\text{batch}} \cdot \max(d, n_L)$ ensures that the vectors used in the dot-product proofs are sufficiently long to accommodate both feature vectors of dimension d and label vectors of dimension n_L; shorter vectors are implicitly zero-padded, which does not affect correctness. Moreover, for the dot-product proofs involving the original dataset, we use auxiliary vectors $\boldsymbol{z}'_{\boldsymbol{X}}$ and $\boldsymbol{z}'_{\boldsymbol{Y}}$ derived z and ψ:

$$\boldsymbol{z}'_{\boldsymbol{X}} \coloneqq \begin{pmatrix} z^{(\psi^{-1}(1)-1)\cdot d}, \dots, z^{\psi^{-1}(1)\cdot d-1}, \dots, \\ z^{(\psi^{-1}(N_{\text{samples}})-1)\cdot d}, \dots, z^{\psi^{-1}(N_{\text{samples}})\cdot d-1} \end{pmatrix},$$

$$\boldsymbol{z}'_{\boldsymbol{Y}} \coloneqq \begin{pmatrix} z^{(\psi^{-1}(1)-1)\cdot n_L}, \dots, z^{\psi^{-1}(1)\cdot n_L-1}, \dots, \\ z^{(\psi^{-1}(N_{\text{samples}})-1)\cdot n_L}, \dots, z^{\psi^{-1}(N_{\text{samples}})\cdot n_L-1} \end{pmatrix}.$$

4.2. Square-Root Commitment Scheme

A major part of (D)NN operations consists of matrix computations; we therefore require efficient ZKPs for such operations. As in other efficient approaches in this area, we rely on sumcheck protocols, given that these are time-optimal for, e.g., matrix multiplication [22]. Ultimately, these require an efficient method for evaluating a (commitment to an) MLE at a random point. The efficiency of these protocols depends primarily on the underlying commitment scheme.

To this end, and in line with our goal of relying on standard cryptographic assumptions while maintaining a balanced prover–verifier trade-off, we adopt the square-root commitment scheme from Hyrax, itself building on work by Groth [23]. This scheme allows us to encode matrices as MLEs and to leverage the resulting efficient method for evaluating the MLE at a random point $\boldsymbol{r} \in \mathbb{F}^\ell$, which has computational cost $O(2^{\ell/2}) = O(\sqrt{n})$ (hence, *square-root*).

Throughout this paper, for clarity of presentation, we commit to a matrix by making a single Pedersen vector commitment per row. Accordingly, all relations are formulated in terms of row-wise commitments to matrices. In practice — and in our implementation (see Section 5) — efficiency gains can be achieved by reshaping the matrix underlying a commitment to be as square as possible prior to committing. Notably, such adaptations require only straightforward modifications to the protocols, without affecting security. For legibility, we furthermore restrict our attention to square matrices in this section.

Algorithm 1 Zero-Knowledge Proof for $\mathcal{R}_{\mathsf{shf\text{-}spl}}$

Public Input: $x = \left(\mathsf{cm}_{\boldsymbol{X}}, \mathsf{cm}_{\boldsymbol{Y}}, \{\mathsf{cm}_{\boldsymbol{X}_i}, \mathsf{cm}_{\boldsymbol{Y}_i}\}_{i\in[N_{\text{batch}}]}, \psi\right)$
Witness: $w = \left(\boldsymbol{X}, r_{\boldsymbol{X}}, \boldsymbol{Y}, r_{\boldsymbol{Y}}, \{r_{\boldsymbol{X}_i}, r_{\boldsymbol{Y}_i}\}_{i\in[N_{\text{batch}}]}\right)$

1: $\mathcal{V}$ randomly selects $z \in \mathbb{F}^*$ and sends this to $\mathcal{P}$.
2: $\mathcal{P}$ computes $t_1 \coloneqq \sum_{j\in[N_{\text{samples}}]} z^{(j-1)\cdot d} \cdot \langle \boldsymbol{z}^d, \boldsymbol{x}_{\psi(j)} \rangle$ and $t_2 \coloneqq \sum_{j\in[N_{\text{samples}}]} z^{N_{\text{samples}}\cdot d+(j-1)\cdot n_L} \cdot \langle \boldsymbol{z}^{n_L}, \boldsymbol{y}_{\psi(j)} \rangle$ and sends its commitments cm_{t_1} and cm_{t_2} to $\mathcal{V}$.
3: $\mathcal{P}$ and $\mathcal{V}$ compute $\mathsf{cm}_{\hat{D}} \coloneqq \Pi_{i\in[N_{\text{batch}}]} (\mathsf{cm}_{\boldsymbol{X}_i})^{z^{S_{\text{batch}}\cdot(i-1)\cdot d}} \cdot (\mathsf{cm}_{\boldsymbol{Y}_i})^{z^{S_{\text{batch}}\cdot(k\cdot d+(i-1)\cdot n_L)}}$.
4: $\mathcal{P}$ and $\mathcal{V}$ execute $\text{proof}_{\log}$-of-dot-product on $(\mathsf{cm}_{\hat{D}}, \boldsymbol{z}^{S_{\text{batch}}\cdot\max(d,n_L)}, \mathsf{cm}_{t_1} \cdot \mathsf{cm}_{t_2})$.
5: $\mathcal{P}$ and $\mathcal{V}$ execute $\text{proof}_{\log}$-of-dot-product on $(\mathsf{cm}_{\boldsymbol{X}}, \boldsymbol{z}'_{\boldsymbol{X}}, \mathsf{cm}_{t_1})$.
6: $\mathcal{P}$ and $\mathcal{V}$ execute $\text{proof}_{\log}$-of-dot-product on $(\mathsf{cm}_{\boldsymbol{Y}}, \boldsymbol{z}'_{\boldsymbol{Y}}, (\mathsf{cm}_{t_2})^{(z^{N_{\text{samples}}\cdot d})^{-1}})$.
7: $\mathcal{V}$ accepts if and only if steps 4, 5, and 6 succeed.

We note that the collections of batch matrix commitments in Section 4.1, namely $\{\mathsf{cm}_{\boldsymbol{X}_i}\}$ and $\{\mathsf{cm}_{\boldsymbol{Y}_i}\}$, also fit within the square-root commitment framework. Here as well, our implementation may apply different transformations of the batch matrices into commitments for efficiency reasons.

4.3. Matrix Multiplication & Hadamard product

To tackle the aforementioned matrix operations using sumchecks, we first present a ZKP for matrix multiplication, allowing $\mathcal{P}$ to convince $\mathcal{V}$ that $\mathcal{P}$ knows matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{n\times n}$ satisfying $\boldsymbol{AB} = \boldsymbol{C}$, given commitments to these matrices. Let $\boldsymbol{A}(i,j)$, $\boldsymbol{B}(i,j)$, and $\boldsymbol{C}(i,j)$ be the multilinear polynomials $\{0,1\}^\ell \times \{0,1\}^\ell \to \mathbb{F}$ such that $\boldsymbol{A}(i,j) = a_{ij}$, $\boldsymbol{B}(i,j) = b_{ij}$, and $\boldsymbol{C}(i,j) = c_{ij}$; as established in Section 2, we assume $n = 2^\ell$. We denote by $\widetilde{\boldsymbol{A}}$, $\widetilde{\boldsymbol{B}}$, and $\widetilde{\boldsymbol{C}}$ their respective MLEs. Later in this section, we show how to extend this protocol to include Hadamard products and how to efficiently aggregate identical checks over different matrices.

Our construction is a zero-knowledge version of Thaler's *optimal* matrix multiplication protocol [22], instantiated using components from Hyrax, in particular we use their proof-of-sum-check construction and their approach for evaluating an MLE on a random point. Whilst other zero-knowledge versions of this protocol exist, we choose these building blocks, to avoid non-standard assumptions, and balance prover and verifier efficiency. The resulting protocol is detailed in Algorithm 2, to better explain the extensions below. We first formalize the corresponding relation.

Definition 4.2 (Matrix Multiplication Relation)**.** *The matrix multiplication relation* $\mathcal{R}_{\mathsf{mat\text{-}mul}}$ *is defined over public inputs*

x and witnesses w. We have $(x; w) \in \mathcal{R}_{\mathsf{mat\text{-}mul}}$ if and only if

$$x = \left(\{\mathsf{cm}_{\boldsymbol{a}_i}\}_{i\in[n]},\ \{\mathsf{cm}_{\boldsymbol{b}_i}\}_{i\in[n]},\ \{\mathsf{cm}_{\boldsymbol{c}_i}\}_{i\in[n]}\right),$$
$$w = \left(\boldsymbol{A}, \{\alpha_i\}_{i\in[n]},\ \boldsymbol{B}, \{\beta_i\}_{i\in[n]},\ \boldsymbol{C}, \{\gamma_i\}_{i\in[n]}\right),$$

such that $\boldsymbol{AB} = \boldsymbol{C}$, and for all $i \in [n]$ $\mathsf{cm}_{\boldsymbol{a}_i} = \mathsf{Com}(\boldsymbol{a}_i, \alpha_i)$, $\mathsf{cm}_{\boldsymbol{b}_i} = \mathsf{Com}(\boldsymbol{b}_i, \beta_i)$, $\mathsf{cm}_{\boldsymbol{c}_i} = \mathsf{Com}(\boldsymbol{c}_i, \gamma_i)$.

At a high level, Algorithm 2 boils down to computing a sumcheck on $\sum_{\boldsymbol{r}_3\in\{0,1\}^\ell} \widetilde{\boldsymbol{A}}(\boldsymbol{r}_1, \boldsymbol{r}_3) \cdot \widetilde{\boldsymbol{B}}(\boldsymbol{r}_3, \boldsymbol{r}_2)$ for given $\boldsymbol{r}_1, \boldsymbol{r}_2$, and verifying that it equals $\widetilde{\boldsymbol{C}}(\boldsymbol{r}_1, \boldsymbol{r}_2)$.

In steps 2 and 8 , $\mathcal{P}$ and $\mathcal{V}$ establish that the commitment $t_{\boldsymbol{C}}$ represents the outcome of evaluating the matrix $\mathbf{C}$ (as given by commitments to the rows in the first step) at $\boldsymbol{r}_1, \boldsymbol{r}_2$. To do this, they engage in Algorithm 6 in Appendix C.2, (following Hyrax [20, Section 6.1]). An analogous computation is performed for matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ in steps 5–7. The proof-of-product in step 9 proves a product relation on Pedersen commitments, as described in, e.g., [20, Figure 5].

The actual verification of the sumcheck relations occurs in step 10. We invoke the proof-of-sumcheck-aggregation described in Algorithm 7, which is taken from Hyrax [20, Figure 1], and slightly restate it to fit our setting. Rather than checking the sumcheck relations round-by-round, the checks are aggregated into a single matrix–vector relation. The first preparations for this aggregation are made in step 3.

4.3.1. Hadamard Product Extension. We also frequently encounter operations of the form $\boldsymbol{C} = (\boldsymbol{A}_1 \circ \boldsymbol{A}_2)\ \boldsymbol{B}$. To express it using a sumcheck protocol, we introduce random vectors $\boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathbb{F}^\ell$. The corresponding sumcheck verifies that $\widetilde{\boldsymbol{C}}(\boldsymbol{r}_1, \boldsymbol{r}_2) = \sum_{\boldsymbol{r}_3\in\{0,1\}^\ell} \widetilde{\boldsymbol{A}_1 \circ \boldsymbol{A}_2}(\boldsymbol{r}_1, \boldsymbol{r}_3)\ \widetilde{\boldsymbol{B}}(\boldsymbol{r}_3, \boldsymbol{r}_2)$.

The key challenge is to efficiently compute the MLE of the Hadamard product. We address this by substituting the expression $\widetilde{\boldsymbol{A}_1 \circ \boldsymbol{A}_2}(\boldsymbol{r}_1, \boldsymbol{r}_3) = \sum_{\boldsymbol{r}_{1,1},\boldsymbol{r}_{1,2}\in\{0,1\}^\ell} \widetilde{\beta}(\boldsymbol{r}_1, \boldsymbol{r}_{1,1}, \boldsymbol{r}_{1,2})\ \widetilde{\boldsymbol{A}}_1(\boldsymbol{r}_{1,1}, \boldsymbol{r}_3)\ \widetilde{\boldsymbol{A}}_2(\boldsymbol{r}_{1,2}, \boldsymbol{r}_3)$, where $\beta\colon \{0,1\}^{3\ell} \to \{0,1\}$ is the indicator function defined by $\beta(x, y, z) = 1$ if and only if $x = y = z$, and $\widetilde{\beta}$ its MLE. This way we maintain multilinearity and correctly evaluate $(\boldsymbol{A_1} \circ \boldsymbol{A_2})(i, j) = \boldsymbol{A_1}(i, j) \cdot \boldsymbol{A_2}(i, j)$ on any $(i, j) \in \{0, 1\}^\ell$.

We choose this substitution over sequentially verifying the two relations, as running two instances of Algorithm 7 for degree-2 polynomials is more expensive than running a single instance (with slightly more rounds) for a degree-3 polynomial. Moreover, the number of calls to Algorithm 6 remains unchanged, and the additional cost of evaluating β is marginal. Algorithms 2 and 7 can be adapted to handle the Hadamard product case with the modifications below.

Modifications to Algorithm 2:

- Use commitments to both $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ (instead of just $\boldsymbol{A}$).
- Step 3: The prover sends 4 coefficients (instead of 3).
- Step 5: Evaluate both $t_{\boldsymbol{A_1}}$ and $t_{\boldsymbol{A_2}}$ (instead of just $t_{\boldsymbol{A}}$).
- Step 6: Execute for both $(\boldsymbol{A_1}, \mathsf{cm}_{t_{\boldsymbol{A_1}}})$ and $(\boldsymbol{A}_2, \mathsf{cm}_{t_{\boldsymbol{A_2}}})$.
- Step 9: Execute an adapted proof-of-product for 3 terms (instead of 2), e.g., two sequential proof-of-products.

Modifications to Algorithm 7:

Algorithm 2 Zero-Knowledge Proof for $\mathcal{R}_{\mathsf{mat\text{-}mul}}$

Public Input: $x = \left(\{\mathsf{cm}_{\boldsymbol{a}_i}\}_{i\in[n]}, \{\mathsf{cm}_{\boldsymbol{b}_i}\}_{i\in[n]}, \{\mathsf{cm}_{\boldsymbol{c}_i}\}_{i\in[n]}\right)$
Witness: $w = \left(\boldsymbol{A}, \{\alpha_i\}_{i\in[n]}, \boldsymbol{B}, \{\beta_i\}_{i\in[n]}, \boldsymbol{C}, \{\gamma_i\}_{i\in[n]}\right)$

1: $\mathcal{V}$ selects random vectors $\boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathbb{F}^\ell$ and sends these to $\mathcal{P}$.
2: $\mathcal{P}$ computes $t_{\boldsymbol{C}} \coloneqq \widetilde{\boldsymbol{C}}(\boldsymbol{r}_1, \boldsymbol{r}_2) \in \mathbb{F}$ and sends its commitment $\mathsf{cm}_{t_{\boldsymbol{C}}}$ to $\mathcal{V}$.
3: **for** $j = 1, \ldots, \ell$ **do**
  $\mathcal{P}$ commits to this sumcheck round's polynomial coefficients $\boldsymbol{c}_j \coloneqq (c_{2,j}, c_{1,j}, c_{0,j})$ using randomness $r_{\boldsymbol{c}_j}$, resulting in $\mathsf{cm}_{\boldsymbol{c}_j}$ and sends this to $\mathcal{V}$.
  $\mathcal{V}$ selects random $r_{3,j} \in \mathbb{F}$ and sends it to $\mathcal{P}$.
4: **end for**
5: $\mathcal{P}$ computes $t_{\boldsymbol{A}} \coloneqq \widetilde{\boldsymbol{A}}(\boldsymbol{r}_1, \boldsymbol{r}_3)$, $t_{\boldsymbol{B}} \coloneqq \widetilde{\boldsymbol{B}}(\boldsymbol{r}_3, \boldsymbol{r}_2)$, and $t_{\boldsymbol{AB}} \coloneqq t_{\boldsymbol{A}} \cdot t_{\boldsymbol{B}}$, where $\boldsymbol{r}_3 \coloneqq (r_{3,1}, \ldots, r_{3,\ell})$. $\mathcal{P}$ also computes the commitments $\mathsf{cm}_{t_{\boldsymbol{A}}}$, $\mathsf{cm}_{t_{\boldsymbol{B}}}$ and $\mathsf{cm}_{t_{\boldsymbol{AB}}}$ and sends these to $\mathcal{V}$.
6: $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 6 on $(\{\mathsf{cm}_{\boldsymbol{a}_i}\}, \mathsf{cm}_{t_{\boldsymbol{A}}}, \boldsymbol{r}_1, \boldsymbol{r}_3)$, to establish $t_{\boldsymbol{A}} = \widetilde{\boldsymbol{A}}(\boldsymbol{r}_1, \boldsymbol{r}_3)$.
7: $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 6 on $(\{\mathsf{cm}_{\boldsymbol{b}_i}\}, \mathsf{cm}_{t_{\boldsymbol{B}}}, \boldsymbol{r}_3, \boldsymbol{r}_2)$, to establish $t_{\boldsymbol{B}} = \widetilde{\boldsymbol{B}}(\boldsymbol{r}_3, \boldsymbol{r}_2)$.
8: $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 6 on $(\{\mathsf{cm}_{\boldsymbol{c}_i}\}, \mathsf{cm}_{t_{\boldsymbol{C}}}, \boldsymbol{r}_1, \boldsymbol{r}_2)$, to establish $t_{\boldsymbol{C}} = \widetilde{\boldsymbol{C}}(\boldsymbol{r}_1, \boldsymbol{r}_2)$.
9: $\mathcal{P}$ and $\mathcal{V}$ execute proof-of-product on $(\mathsf{cm}_{t_{\boldsymbol{A}}}, \mathsf{cm}_{t_{\boldsymbol{B}}}, \mathsf{cm}_{t_{\boldsymbol{AB}}})$, to establish $t_{\boldsymbol{AB}} = t_{\boldsymbol{A}} t_{\boldsymbol{B}}$.
10: $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 7 on $\left(\{\mathsf{cm}_{\boldsymbol{c}_j}\}, \mathsf{cm}_{t_{\boldsymbol{C}}}, \mathsf{cm}_{t_{\boldsymbol{AB}}}, \boldsymbol{r}_3\right)$, to establish that the sumcheck relations hold.
11: $\mathcal{V}$ accepts if and only if steps 6, 7, 8, 9, and 10 succeed.

- Sum-check polynomials of degree 3 (instead of 2), resulting in:
  - Vector $\boldsymbol{d}$ contains 4ℓ;
  - Each $\mathsf{cm}_{\boldsymbol{d}_j}$ commits to 4 terms (instead of 3).

4.3.2. Amortizing Checks over Multiple Matrices. Due to the iterative nature of training, as well as the structure of (D)NNs, a small set of matrix operations is executed on different matrices each round or layer. Using sumchecks, we can amortize such checks to improve efficiency. In particular, we amortize all checks that follow the same relation over the same matrices, where the only differences are the layer and round indices.

This can be achieved by replacing each function (and its corresponding MLE) $\boldsymbol{A}(i, j)$ with a function $\boldsymbol{A}(k, i, j)$, where $\boldsymbol{A}(i' \cdot L + \ell, i, j) = (\boldsymbol{A}_{i'}^{(\ell)})_{ij}$. This transformation renders the resulting sumcheck formulations non-linear; however, linearity can be restored by introducing an additional indicator function β, analogous to the construction used in the Hadamard product extension (cf. Section 4.3.1).

While this approach increases both the number of rounds and the degree of the polynomials in the sumcheck protocol, the resulting overhead is significantly smaller than the cost of executing all checks independently. The exact sumcheck formulations used in FALAFEL are described in Appendix D.

4.4. Scaling & Activation Function

A prover can now convince a verifier that it correctly constructed batch matrices from the audited dataset and has all the tools to attest to correct execution of the varying matrix

operations in the training process. However, since we will be working with fixed-point representations of our datasets, we also require mechanisms to prove correct rescaling after multiplication. We also show how to adapt these proofs to attest to correct evaluation of the ReLU activation function.

Our final core component is an adaptation of the aggregated logarithmic range proofs from Bulletproofs [13, Section 4.3] to prove correctness of scaling. While rangeproofs are presented for commitments to individual values, they can be readily extended to work with vector commitments to the rows of a matrix. Moreover, where rangeproofs only verify that a value lies within a given range ($b \in [0, 2^q]$), we verify that one value equals the truncated version of another one: $b \in [0, 2^q]$, $a \in [0, 2^s]$, $a = b \gg s$, where "$\gg s$" denotes a binary right-shift by s bits. Subsequently, we show how to extend this proof of scaling in order to prove correctness of the activation function evaluation. Below, we adopt Python-style slice notation, i.e., $\boldsymbol{v}_{[a:b]} \coloneqq (v_i)_{i=a}^{b-1}$.

4.4.1. Proof of Scaling. First, fix $q \in \mathbb{Z}_{\geq 0}$, the total bit-precision with which we encode all real numbers in FALAFEL. Then, we encode a real number x as a fixed-point integer under the constraint $|x| < 2^{q-1}$. In our setting, each real number $x \in \mathbb{R}$ is represented in the finite field $\mathbb{F}$ by the encoded value $\widetilde{x} = \lfloor 2^s x \rfloor$ with s-bit precision, so decoding is given by $x \approx 2^{-s}\widetilde{x}$. For two encoded values $\widetilde{x}, \widetilde{y} \in \mathbb{F}$, their product satisfies $\widetilde{x}\widetilde{y} = \lfloor 2^s x \rfloor \lfloor 2^s y \rfloor = \lfloor 2^{2s} xy \rfloor$. Thus, multiplication in $\mathbb{F}$ unintentionally increases the scaling factor from 2^s to 2^{2s}, i.e., we need to rescale.

For rescaling, it is convenient to work with nonnegative values. Since $|x| < 2^{q-1}$, therefore we shift the values in $\boldsymbol{A}$ by 2^{q-1-s} and in $\boldsymbol{B}$ by 2^{q-1}. For legibility, in this section and Algorithm 3 we write $\boldsymbol{B}$ or $\boldsymbol{A}$, for what actually is $\boldsymbol{B} + \boldsymbol{1}_{n\times n} \cdot 2^{q-1}$ respectively $\boldsymbol{A} + \boldsymbol{1}_{n\times n} \cdot 2^{q-1-s}$.

More concretely, $\mathcal{P}$ wishes to convince $\mathcal{V}$ that A is a rescaling of B by a factor 2^s, i.e., for given commitments, it knows the corresponding matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{F}^{n\times n}$ for which $a_{ij} = b_{ij} \gg s$ should hold for all $i, j \in [n]$. Formally, this is captured in the following relation.

Definition 4.3 (Scaling Relation)**.** *The scaling relation $\mathcal{R}_{\mathsf{scaling}}$ is defined over public inputs x and witnesses w. We have $(x; w) \in \mathcal{R}_{\mathsf{scaling}}$ if and only if*

$$x = \left(\{\mathsf{cm}_{\boldsymbol{a}_j}\}_{j\in[n]}, \{\mathsf{cm}_{\boldsymbol{b}_j}\}_{j\in[n]}, s\right),$$
$$w = \left(\boldsymbol{A}, \{\alpha_i\}_{i\in[n]}, \boldsymbol{B}, \{\beta_i\}_{i\in[n]}\right),$$

such that $b_{ij} - 2^s a_{ij} \in [0, 2^s)$ for all $i, j \in [n]$ and for all $i \in [n]$: $\mathsf{cm}_{\boldsymbol{a}_i} = \mathsf{Com}(\boldsymbol{a}_i, \alpha_i)$, $\mathsf{cm}_{\boldsymbol{b}_i} = \mathsf{Com}(\boldsymbol{b}_i, \beta_i)$.

The ZKP for this relation is presented in Algorithm 3. Whereas Bulletproofs describe an efficient aggregation mechanism for multiple rangeproofs, given an individual commitment to each value, we work with matrices of values, packaged in vector commitments to the rows. We overcome this difference by modifying the set-up steps 1–3, to concatenate the bits of all matrix entries. We then add an invocation of the proof$_{\log}$-of-dot-product in step 4, to show that this combination of the matrix entries into linear combinations is consistent with the rows of $\boldsymbol{A}$ and $\boldsymbol{B}$.

Algorithm 3 Zero-Knowledge Proof for $\mathcal{R}_{\mathsf{scaling}}$

Public Parameters: generators $g, h \in \mathbb{G}$; generators $\boldsymbol{g}, \boldsymbol{h} \in \mathbb{G}^{q\cdot n^2}$
Public Input: $x = \left(\{\mathsf{cm}_{\boldsymbol{a}_j}\}_{j\in[n]}, \{\mathsf{cm}_{\boldsymbol{b}_j}\}_{j\in[n]}, s\right)$
Witness: $w = \left(\boldsymbol{A}, \{\alpha_i\}_{i\in[n]}\ \boldsymbol{B}, \{\beta_i\}_{i\in[n]}\right)$

1: $\mathcal{P}$ defines $\boldsymbol{a}_L$ to be the vector containing the concatenation of all bits of $\{b_{ij}\}_{i,j\in[n]}$ and sets $\boldsymbol{a}_R = \boldsymbol{a}_L - \boldsymbol{1}^{q\cdot n^2}$. Then randomly samples $\alpha \in \mathbb{F}$, and defines the commitment $\mathsf{cm}_{\boldsymbol{a}} \coloneqq \boldsymbol{g}^{\boldsymbol{a}_L}\boldsymbol{h}^{\boldsymbol{a}_R}h^{\alpha}$. $\mathcal{P}$ also randomly selects $\boldsymbol{s}_L, \boldsymbol{s}_R \in \mathbb{F}^{qn^2}$, $\rho \in \mathbb{F}$, defines $\mathsf{cm}_{\boldsymbol{s}} \coloneqq \boldsymbol{g}^{\boldsymbol{s}_L}\boldsymbol{h}^{\boldsymbol{s}_R}h^{\rho}$, and sends $\mathsf{cm}_{\boldsymbol{a}}$ and $\mathsf{cm}_{\boldsymbol{s}}$ to $\mathcal{V}$.
2: $\mathcal{V}$ randomly selects $z \in \mathbb{F}^*$ and sends this to $\mathcal{P}$.
3: $\mathcal{P}$ computes $v = \sum_{i=1}^{n} z^{(i-1)n} \sum_{j=1}^{n} z^{j-1} b_{ij} + z^{n^2} \sum_{i=1}^{n} z^{(i-1)n} \sum_{j=1}^{n} z^{j-1} a_{ij}$, randomly samples $\gamma \in \mathbb{F}$, computes $\mathsf{cm}_v = g^v h^{\gamma}$ and shares cm_v with $\mathcal{V}$.
4: $\mathcal{P}$ and $\mathcal{V}$ execute proof$_{\log}$-of-dot-product on $((\prod_{j=1}^{n} \mathsf{cm}_{\boldsymbol{b}_j}^{z^{(j-1)n}}) \cdot (\prod_{j=1}^{n} \mathsf{cm}_{\boldsymbol{a}_j}^{z^{(j-1)n}})^{z^{n^2}}, \boldsymbol{z}^n, \mathsf{cm}_v)$.
5: $\mathcal{V}$ randomly selects $y \in \mathbb{F}^*$ and sends this to $\mathcal{P}$.
6: $\mathcal{P}$ computes the following polynomials:
- $l(X) \coloneqq \boldsymbol{a}_L - z \cdot \boldsymbol{1}^{q\cdot n^2} + \boldsymbol{s}_L \cdot X$;
- $r(X) \coloneqq \boldsymbol{y}^{q\cdot n^2} \circ (\boldsymbol{a}_R + z \cdot \boldsymbol{1}^{q\cdot n^2} + \boldsymbol{s}_R \cdot X)$
  $+ \sum_{j=1}^{n^2} z^{1+j} \cdot \boldsymbol{0}^{(j-1)\cdot q} \,||\, \boldsymbol{2}^q \,||\, \boldsymbol{0}^{(n^2-j)\cdot q}$
  $+ \sum_{j=1}^{n^2} z^{n^2+1+j} \cdot \boldsymbol{0}^{(j-1)\cdot q} \,||\, \boldsymbol{0}^{s} \,||\, \boldsymbol{2}^{q-s} \,||\, \boldsymbol{0}^{(n^2-j)\cdot q}$;
- $t(X) \coloneqq \langle l(X), r(X)\rangle = \sum_{i=0}^{2} t_i \cdot X^i$.

Next, $\mathcal{P}$ randomly samples $\tau_1, \tau_2 \in \mathbb{F}$ and sends the commitments $\mathsf{cm}_{t_i} = g^{t_i} h^{\tau_i}, i \in \{1, 2\}$ to $\mathcal{V}$.
7: $\mathcal{V}$ randomly selects $x \in \mathbb{F}^*$ and sends it to $\mathcal{P}$.
8: $\mathcal{P}$ computes $\tau_x \coloneqq \tau_1 \cdot x + \tau_2 \cdot x^2 + z^2 \cdot \gamma$, $\mu \coloneqq \alpha + \rho \cdot x$, $\boldsymbol{l} \coloneqq l(x)$, $\boldsymbol{r} \coloneqq r(x)$, $\hat{t} \coloneqq \langle \boldsymbol{l}, \boldsymbol{r}\rangle$, and sends τ_x, μ, and $\hat{t}$ to $\mathcal{V}$.
9: $\mathcal{P}$ and $\mathcal{V}$ compute $h'_j = h_j^{y^{-j+1}}$ for all $j \in [1, qn^2]$, where h_j is the j-th component of $\boldsymbol{h}$. They also compute
- $Q_1 = \prod_{j=1}^{n^2} (\boldsymbol{h}'_{[(j-1)\cdot q : j\cdot q - 1]})^{z^{1+j}\cdot \boldsymbol{2}^q}$;
- $Q_2 = \prod_{j=1}^{n^2} (\boldsymbol{h}'_{[(j-1)\cdot q + s : j\cdot q]})^{z^{n^2+1+j}\cdot \boldsymbol{2}^{q-s}}$;
- $P = \mathsf{cm}_{\boldsymbol{a}} \cdot (\mathsf{cm}_{\boldsymbol{s}})^x \cdot \boldsymbol{g}^{-z} \cdot \boldsymbol{h}'^{z\cdot \boldsymbol{y}^{q\cdot n^2}} \cdot Q_1 \cdot Q_2$.

10: $\mathcal{V}$ evaluates $\delta(y, z) = (z - z^2) \cdot \langle \boldsymbol{1}^{q\cdot n^2}, \boldsymbol{y}^{q\cdot n^2}\rangle - \sum_{j=1}^{n^2} z^{2+j} \cdot \langle \boldsymbol{1}^q, \boldsymbol{2}^q\rangle - \sum_{j=1}^{n^2} z^{2+n^2+j} \cdot \langle \boldsymbol{1}^{q-s}, \boldsymbol{2}^{q-s}\rangle$ and checks that $g^{\hat{t}} h^{\tau_x} = g^{\delta(y,z)} \cdot (\mathsf{cm}_v)^{z^2} \cdot (\mathsf{cm}_{t_1})^x \cdot (\mathsf{cm}_{t_2})^{x^2}$.
11: $\mathcal{P}$ and $\mathcal{V}$ execute Bulletproofs' Inner Product Argument on $(\boldsymbol{g}, \boldsymbol{h}', P \cdot h^{-\mu}, \hat{t})$.
12: $\mathcal{V}$ accepts if and only if steps 4, 10, and 11 succeed.

With these adaptations we can show $b_{ij} \in [0, 2^q)$ for all b_{ij} in $\boldsymbol{B}$. To show that each a_{ij} corresponds to the appropriately shifted bits of b_{ij}, we introduce the new third additive term in the definition of $r(X)$ in step 6, and update the corresponding computations and checks in the steps thereafter analogously. This approach only adds minor overhead compared to a standard aggregated range proof: the number of additional group operations in step 4 is relatively small, and the additional field operations are marginal.

4.4.2. Proof of Activation Function. Given a pre-activation matrix $\boldsymbol{T}$, we compute the post-activation matrix $\boldsymbol{U}$ by applying $\mathrm{ReLU}(\cdot)$ entry-wise. For this purpose, define a matrix $\boldsymbol{T}'$ by setting $t'_{ij} = 1$ if $t_{ij} > 0$ and 0 otherwise. Then $\boldsymbol{T}'$ applies the ReLU mask, i.e., $\boldsymbol{U} = \boldsymbol{T}' \circ \boldsymbol{T}$.

We observe that this matrix $\boldsymbol{T}'$ can be constructed with only minor modifications to Algorithm 3. The key change is to extend the definition of the polynomial $r(X)$ in step 6 by adding the following expression as a fourth additive term: $\sum_{j=1}^{n^2} z^{2n^2+1+j} \cdot \mathbf{0}^{(j-1)\cdot q} \,||\, \mathbf{0}^{q-1} \,||\, 1 \,||\, \mathbf{0}^{(n^2-j)\cdot q}$. This additional term extracts the sign bit of each entry; since $\boldsymbol{B}$ was shifted from the interval $[-2^{q-1}, 2^{q-1})$ to $[0, 2^q)$, each entry's most significant bit corresponds to its sign.

Several further adjustments to Algorithm 3 are required to support this extension. As public input we now also include commitments to the rows $\{\mathsf{cm}_{\boldsymbol{t}'_j}\}_{j\in[n]}$ of a third matrix $\boldsymbol{T}' \coloneqq \mathrm{ReLU}(\boldsymbol{B})$. The definition of v in step 3 receives an additional additive component to incorporate the values c_{ij}. Likewise, steps 4, 9, and 10 are updated in a straightforward manner to reflect the presence of this new component.

We argue that this proof of activation function is efficient. The extension adds overhead that is similar in nature to that added by the proof of scaling (when compared to the base protocol), i.e., a relatively small number of additional group operations. The additional field operations are marginal.

5. The FALAFEL Protocol

Having introduced all components a client needs to attest to during its local training phrase, we present the full FALAFEL protocol by discussing the offline, training (client- and server-side), and usage phase. But, before doing so, we introduce the notation for all commitments used in FALAFEL.

Commitments. We slightly abuse notation when referring to commitments. Concretely, when writing, $\mathsf{cm}_{\boldsymbol{T}}$, we refer to the collection of commitments to *all* matrices $\{\boldsymbol{T}_i^{(\ell)}\}_{i,\ell}$ appearing in the protocol. The same convention is used for $\mathsf{cm}_{\boldsymbol{T}'}, \mathsf{cm}_{\hat{\boldsymbol{T}}}, \mathsf{cm}_{\boldsymbol{R}}, \mathsf{cm}_{\hat{\boldsymbol{R}}}, \mathsf{cm}_{\overline{\boldsymbol{G}}}, \mathsf{cm}_{\hat{\boldsymbol{G}}}$. Likewise, $\mathsf{cm}_{\boldsymbol{\mathcal{W}}}$ denotes the collection of commitments to all weight matrices.

Furthermore, $\mathsf{cm}_{\boldsymbol{B}}$ denotes two collections of commitments to the individual mini-batches derived from D: one collection $\mathsf{cm}_{\boldsymbol{X}}$ for the inputs and one $\mathsf{cm}_{\boldsymbol{Y}}$ for the labels. Finally, $\mathsf{cm}_{\mathbf{trace}}$ denotes the collection of all commitments appearing in the training transcript (cf. Theorem 5.1), with $\mathbf{trace}$ denoting the underlying matrices and $\{r_*\}$ the corresponding commitments' randomness.

We commit to all matrices of one type using a single commitment, as follows. We take $\{\boldsymbol{T}_i^{(\ell)}\}_{i,\ell}$ and $\mathsf{cm}_{\boldsymbol{T}}$ as a leading example. First, for each iteration i and each $\ell \in \{0, L+1\}$, we add an all-zero matrix of the appropriate dimensions and assign each matrix a unique index $m \coloneqq i \cdot (L+2) + \ell$. Next, we transform the resulting collection of matrices into a vector $\boldsymbol{t}$ by assigning the entry in the c-th column of the r-th row of the m-th matrix to index $m + r \cdot \lceil \log(M) \rceil + c \cdot \lceil \log(M+R) \rceil$, where M denotes the total number of matrices and R the maximum number of rows among them. Unassigned entries of $\boldsymbol{t}$ are filled using implicit zero-padding. Finally, we commit to $\boldsymbol{t}$ using the square-root commitment scheme of [20]. That is, we encode $\boldsymbol{t}$ as a matrix $\boldsymbol{T}$ whose column-major order equals $\boldsymbol{t}$, and define a single row-commitment for each row of $\boldsymbol{T}$. In general, we choose $\boldsymbol{T}$ to be as square as possible.

Lastly, cm_D denotes two regular Pedersen vector commitments to the full dataset $D = (\boldsymbol{X}, \boldsymbol{Y})$, namely one commitment to the inputs and one to the labels.

5.1. Offline Phase

The auditor signs a compact commitment cm_{D_k} to each client P_k's dataset D_k (for $k \in [K]$), ensuring dataset authenticity. All clients and the server S agree on the public NN architecture NN (including initial weights $\boldsymbol{\mathcal{W}}_{\mathrm{glob},0}$, learning rate η, precision s), and public parameters pp.

5.2. Training Phase — Overview

The training phase (Algorithm 4) begins with each client P_k broadcasting their dataset commitment cm_{D_k} together with the auditor's signature $\sigma_{\mathsf{cm}_{D_k}}$. All clients and S verify these signatures, aborting if any fail. The clients and S then execute N_{rounds} training rounds.

In every round r, each P_k locally trains for E epochs on D_k, starting with $\boldsymbol{\mathcal{W}}_{\mathrm{glob},r-1}$ and computing both the updated weights $\boldsymbol{\mathcal{W}}_{k,r}$, as well as a proof $\pi_{k,r}$ attesting to correct local training. The weights $\boldsymbol{\mathcal{W}}_{k,r}$ are shared only with S and the proof $\pi_{k,r}$ is broadcast publicly. $\pi_{k,r}$ also includes this round's commitments it proves correctness of.

Then, S proceeds by verifying all proofs $\{\pi_{k,r}\}_{k\in[K]}$, and aggregates the received weights into $\boldsymbol{\mathcal{W}}_{\mathrm{glob},r}$ while also computing a proof of correct aggregation $\pi_{\mathrm{glob},r}$. The server shares $\boldsymbol{\mathcal{W}}_{\mathrm{glob},r}$ with all clients directly, and broadcasts $\pi_{\mathrm{glob},r}$. Each P_k now verifies $\pi_{\mathrm{glob},r}$ and proceeds to round $r+1$. This process continues until some convergence criterion is met in round N_{rounds} at which point the final weights $\boldsymbol{\mathcal{W}}_{\mathrm{final}} \coloneqq \boldsymbol{\mathcal{W}}_{\mathrm{glob},N_{\mathrm{rounds}}}$ are broadcast.

All proofs can be verified using the commitments alone. The openings of weight commitments are only sent for update and aggregation purposes, and are checked for consistency against the corresponding proof.

5.3. Training Phase — Client-Side

The client-side protocol for proving correctness of local training round r corresponds to steps 9–12 of Algorithm 4. P_k performs training using mini-batch gradient descent. Our only deviations from the standard ML approach are the explicit construction of the ReLU masks $\boldsymbol{T}_i'^{(\ell)}$, and the inclusion of intermediate scaling steps to ensure correct fixed-point arithmetic throughout the computation. After completing this local training procedure (see Appendix C.4), P_k generates the ZKP π_k attesting to the correctness of training (cf. Theorem 5.1) using Algorithm 5, defining $\boldsymbol{\mathcal{W}}_0 \coloneqq \boldsymbol{\mathcal{W}}_{\mathrm{glob},r-1}$.

Definition 5.1 (Local Training Relation)**.** *The local training relation $\mathcal{R}_{\mathsf{train}}$ is defined over public inputs x and witnesses w. We have $(x; w) \in \mathcal{R}_{\mathsf{train}}$ if and only if*

$$x = \left(\mathsf{cm}_{\mathbf{trace}}, \mathsf{cm}_{\boldsymbol{\mathcal{W}}_0}, \mathsf{cm}_D, s, \eta, \psi\right),$$

Algorithm 4 FALAFEL Training Phase

Public input: NN, $(\boldsymbol{\mathcal{W}}_{\text{glob},0}, r_{\boldsymbol{\mathcal{W}}_{\text{glob},0}})$, η, s, E, and N_{rounds}
Client P_k's input: D_k, cm_{D_k}, $\sigma_{\mathsf{cm}_{D_k}}$

Each P_k broadcasts cm_{D_k} and $\sigma_{\mathsf{cm}_{D_k}}$.
Each P_k and S verify received signatures.
for $r = 1, \ldots, N_{\text{rounds}}$ **do**
  for $k \in [K]$, let P_k **do**
    Receive $\pi_{h,r-1}$ for all $h \in [K] \setminus \{k\}$ and verify.
    Receive $o_{\text{glob},r} := (\boldsymbol{\mathcal{W}}_{\text{glob},r}, r_{\boldsymbol{\mathcal{W}}_{\text{glob},r}})$ from S.
    Verify $\pi_{\text{glob},r-1}$ that was broadcast by S.
    Check that $o_{\text{glob},r}$ opens $\mathsf{cm}_{\mathcal{W}_{\text{glob}}}$ in $\pi_{\text{glob},r-1}$
    Execute local training for E epochs on $\boldsymbol{\mathcal{W}}_{\text{glob},r-1}$.
    Obtain $\mathbf{trace}_{k,r}$ and new weights $\boldsymbol{\mathcal{W}}_{k,r}$.
    Commit to $\mathbf{trace}_{k,r}$ and to new weights.
    Generate $\pi_{k,r}$ using the FS-transform of Algorithm 5.
    Broadcast $\pi_{k,r}$ and send $(\boldsymbol{\mathcal{W}}_{k,r}, r_{\boldsymbol{\mathcal{W}}_{k,r}})$ to S.
  end for
  let S **do**
    Receive $o_{k,r} := (\boldsymbol{\mathcal{W}}_{k,r}, r_{\boldsymbol{\mathcal{W}}_{k,r}})$ from P_k for $k \in [K]$.
    Verify all $\pi_{k,r}$ and check that $o_{k,r}$ opens $\mathsf{cm}_{\boldsymbol{\mathcal{W}}}$ in $\pi_{k,r}$.
    Aggregate $\{\boldsymbol{\mathcal{W}}_{k,r}\}_{k\in[K]}$ into $\mathcal{W}_{\text{glob},r}$.
    Generate $\pi_{\text{glob},r}$ and broadcast.
    Send $(\boldsymbol{\mathcal{W}}_{\text{glob},r}, r_{\boldsymbol{\mathcal{W}}_{\text{glob},r}})$ to P_k for $k \in [K]$.
end for

$$w = \big(\mathbf{trace}, \boldsymbol{\mathcal{W}}_0, \boldsymbol{\mathcal{W}}_{N_{\text{batch}}\cdot E}, D, \{r_*\}\big),$$

where $\mathbf{trace}$ *consists of the following matrices*

$$\begin{aligned}\mathbf{trace} = \big(&\{\boldsymbol{T}_i'^{(\ell)}, \hat{\boldsymbol{T}}_i^{(\ell)}, \boldsymbol{T}_i^{(\ell)}, \overline{\boldsymbol{G}}_i^{(\ell)}, \hat{\boldsymbol{R}}_i^{(\ell+1)},\\ &\boldsymbol{R}_i^{(\ell+1)}, \hat{\boldsymbol{G}}_i^{(\ell)}\}_{i\in[N_{\text{batch}}\cdot E],\, \ell\in[L]},\\ &\{B_i\}_{i\in[N_{\text{batch}}]},\ \{\boldsymbol{U}_i^{(L)}\}_{i\in[N_{\text{batch}}\cdot E]}\big).\end{aligned}$$

and such that the following conditions hold:

1) ***Commitment openings.*** *All commitments appearing in x correctly open to the corresponding witness values in w using the supplied randomness $\{r_*\}$.*
2) ***Batch formation.*** *The batches $\{B_i\}_{i\in[N_{\text{batch}}]}$ are obtained by shuffling dataset D according to permutation ψ and splitting the result into k equal-size mini-batches.*
3) ***Correctness of training.*** *$(\mathbf{trace}, \boldsymbol{\mathcal{W}}_0, \boldsymbol{\mathcal{W}}_{N_{\text{batch}}\cdot E})$ satisfies the forward pass, backward pass, and weight-update equations of Algorithm 8 (Appendix C.4) with scaling factor s and learning rate η.*
4) ***Boundary conditions.*** *The initial and final model weights in the training transcript coincide with the committed values $\boldsymbol{\mathcal{W}}_0$ and $\boldsymbol{\mathcal{W}}_{N_{\text{batch}}\cdot E}$, respectively.*

At a high level, Algorithm 5 proceeds by sequentially proving *Batch Correctness*, *Forward Pass*, *Backward Pass*, and *Weight Update*. Most steps are straightforward applications of the components from Section 4. Some details are more subtle and are discussed in more detail below. The exact sumcheck relations referred to in steps 2, 4, and 5 are derived in Appendix D following Section 4.3.

Combining Polynomial Evaluations. Each execution of the sumcheck protocols in steps 2, 4, and 5 of Algorithm 5 invokes multiple sub-calls to Algorithm 6, which evaluates the MLE of a matrix on a random point. Several of these

Algorithm 5 Zero-Knowledge Proof for $\mathcal{R}_{\mathsf{train}}$

Public Input: $x = \big(\mathsf{cm}_{\mathbf{trace}}, \mathsf{cm}_{\boldsymbol{\mathcal{W}}_0}, \mathsf{cm}_D, s, \eta, \psi\big)$
Witness: $w = \big(\mathbf{trace}, \boldsymbol{\mathcal{W}}_0, \boldsymbol{\mathcal{W}}_{N_{\text{batch}}\cdot E} D, \{r_*\}\big)$

1: *Batch Correctness:* $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 1 on public input $(\mathsf{cm}_D, \mathsf{cm}_{\boldsymbol{B}}, \psi)$.
2: *Forward Pass (a):* $\mathcal{P}$ and $\mathcal{V}$ execute an adaptation of Algorithm 2 — postponing calls to Algorithm 6 — for $\boldsymbol{T}_i^{(\ell)} = \boldsymbol{W}_{i-1}^{(\ell)} \cdot (\boldsymbol{T}_i'^{(\ell-1)} \circ \hat{\boldsymbol{T}}_i^{(\ell-1)})$, with sumcheck relation (D.1). Here, for all $i \in 1, \ldots, N_{\text{batch}} \cdot E$, we define $\boldsymbol{T}_i^{(L+1)} = \boldsymbol{T}_i'^{(L)} \circ \hat{\boldsymbol{T}}_i^{(L)}$, $\boldsymbol{W}_{i-1}^{(L+1)} = \boldsymbol{I}$, $\hat{\boldsymbol{T}}_i^{(0)} = \boldsymbol{X}_i^{(0)}$ and $\boldsymbol{T}_i'^{(0)} = \mathbf{1}$.
3: *Forward Pass (b):* $\mathcal{P}$ and $\mathcal{V}$ execute the proof-of-activation-function (cf. Section 4.4.2), which includes a proof-of-scaling, on public input $(\mathsf{cm}_{\hat{\boldsymbol{T}}}, \mathsf{cm}_{\boldsymbol{T}}, \mathsf{cm}_{\boldsymbol{C}}, s)$.
4: *Backward Pass (a):* $\mathcal{P}$ and $\mathcal{V}$ execute an adaptation of Algorithm 2 — postponing calls to Algorithm 6 — for $\overline{\boldsymbol{G}}_i^{(\ell)} = (\hat{\boldsymbol{R}}_i^{(\ell+1)} \circ \boldsymbol{T}_i'^{(\ell)}) \cdot (\boldsymbol{T}_i'^{(\ell)} \circ \hat{\boldsymbol{T}}_i^{(\ell-1)})^\top$, with sumcheck relation (D.2). Here, for all $i \in 1, \ldots, N_{\text{batch}} \cdot E$, we define $\hat{\boldsymbol{T}}_i^{(0)} = X_{(i-1) \pmod{N_{\text{batch}}}+1}$ and $\boldsymbol{T}_i'^{(0)} = \mathbf{1}$. Additionally, we set the elements of the first row of $\hat{\boldsymbol{T}}_i^{(\ell)}$ equal to 1, for all $i = 1, \ldots, N_{\text{batch}} \cdot E$ and $\ell = 0, \ldots, L-1$.
5: *Backward Pass (b):* $\mathcal{P}$ and $\mathcal{V}$ execute an adaptation of Algorithm 2 — postponing calls to Algorithm 6 — for $\boldsymbol{R}_i^{(\ell)} = (\boldsymbol{W}_{i-1}^{(\ell)})^\top \cdot (\hat{\boldsymbol{R}}_i^{(\ell+1)} \circ \boldsymbol{T}_i'^{(\ell)})$, with sumcheck relation (D.3). Here, for all $i \in 1, \ldots, N_{\text{batch}} \cdot E$, we define $\boldsymbol{W}_{i-1}^{(L+1)} = s \cdot \boldsymbol{I}$, $\boldsymbol{R}_i^{(L+1)} = \mathbf{0}$, $\boldsymbol{W}_{i-1}^{(L+1)} = s \cdot \boldsymbol{I}$, and $\boldsymbol{T}_i'^{(0)} = \mathbf{1}$.
6: *Backward Pass (c):* $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 3 on public input $(\mathsf{cm}_{\hat{\boldsymbol{R}}}, \mathsf{cm}_{\boldsymbol{R}}, s)$.
7: *Backward Pass (d):* $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 3 on public input $(\mathsf{cm}_{\hat{\boldsymbol{R}}^{(L+1)}}, (\mathsf{cm}_{\boldsymbol{U}^{(L)}} \cdot (\mathsf{cm}_{\boldsymbol{Y}})^{-1})^{\lfloor 2\cdot 2^s / S_{\text{batch}} \rfloor}, s)$.
8: *Weight update (a):* $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 3 on public input $(\mathsf{cm}_{\hat{\boldsymbol{G}}}, (\mathsf{cm}_{\overline{\boldsymbol{G}}})^{\lfloor 2^s \eta \rfloor}, 2s)$.
9: *Weight update (b):* $\mathcal{P}$ and $\mathcal{V}$ iteratively compute for all $i = 1, \ldots, N_{\text{batch}} \cdot E$, $\mathsf{cm}_{\boldsymbol{W}_i} = \mathsf{cm}_{\boldsymbol{W}_{i-1}} \cdot \mathsf{cm}_{\hat{\boldsymbol{G}}_i}$.
10: *Poly. evals:* $\mathcal{P}$ and $\mathcal{V}$ execute the postponed calls to Algorithm 6 from steps 2, 4, 5, following Section 5.3.
11: $\mathcal{V}$ accepts if and only if all preceding proofs succeed.

evaluations concern the same underlying matrix and can therefore be combined. Computing this combination only requires a small number of group operations, compared to those needed for Algorithm 6. To exploit this, we postpone all invocations of Algorithm 6 and batch them per matrix type (e.g., $\boldsymbol{W}$, $\hat{\boldsymbol{R}}$, or $\boldsymbol{R}$). If an MLE $\widetilde{\boldsymbol{M}}$ of a matrix $\boldsymbol{M}$ is evaluated at multiple random points $\boldsymbol{r}_i$, we reduce these checks to a single evaluation at a freshly sampled point $\boldsymbol{r}^*$. As a result, Algorithm 6 is executed only once per matrix. This aggregation technique is commonly used in prior work to reduce multiple evaluations of the same MLE to a single one, e.g., as in [20], [22], [24].

To combine d evaluations of $\widetilde{\boldsymbol{M}}(r_1, \ldots, r_{\log(n)})$, let $\phi : \mathbb{F} \to \mathbb{F}^{\log(n)}$ be a degree-$(d-1)$ curve passing through the points $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_d$, and define $f_\phi(t)$ as the restriction of $\widetilde{\boldsymbol{M}}$ to this curve, e.g., we instantiate ϕ such that $\phi(i-1) = \boldsymbol{r}_i$ for all $i \in [d]$. $\mathcal{P}$ then sends the univariate polynomial $f_\phi^*(t) = \widetilde{\boldsymbol{M}}(\phi(t))$, which has degree $(d-1)\log(n)$, to $\mathcal{V}$, who accepts if and only if $f_\phi^*(i-1) = \widetilde{\boldsymbol{M}}(r_i)$ for all $i \in [d]$ and $f_\phi^*(r^*) = \widetilde{\boldsymbol{M}}(\phi(t^*))$. The former equalities can

be efficiently checked using the homomorphic properties of Pedersen commitments together with a proof-of-equality protocol, e.g., following [20]. The latter equality is verified by a single invocation of Algorithm 6, thereby reducing d evaluations to one with minimal overhead.

Weight Updates. In steps 8 and 9 of Algorithm 5 P_k updates its model parameters using gradient descent. That is, for each $\ell \in [L]$, P_k does the following. First, they compute the scaled gradient $\eta \cdot \overline{\boldsymbol{G}}_i^{(\ell)}$ and rescale it to obtain $\hat{\boldsymbol{G}}_i^{(\ell)}$. The updated weights are then given by $\boldsymbol{W}_i^{(\ell)} = \boldsymbol{W}_{i-1}^{(\ell)} - \hat{\boldsymbol{G}}_i^{(\ell)}$. To prove correctness of this update, we observe that commitments to the rows of $\eta \cdot \overline{\boldsymbol{G}}_i^{(\ell)}$ are obtained by exponentiating the commitments to the rows of $\overline{\boldsymbol{G}}_i^{(\ell)}$ by η. A proof for the rescaling step is generated using Algorithm 3, while the final subtraction to obtain is applied directly at the level of commitments. This homomorphism also allows S and all P_k to compute commitments to the final and intermediate weights, where necessary.

Combining Commitments & Adding Public Values. In steps 2, 4, and 5 of Algorithm 5, it is sometimes necessary to evaluate an MLE $\widetilde{\boldsymbol{M}}$ at a random point when the underlying values are distributed over multiple commitments, in order to verify that the result equals the claimed value $t_{\boldsymbol{M}}$. As an example, in step 4 all matrices $\hat{\boldsymbol{T}}_i^{(0)}$ are instantiated as batch matrices. These batch matrices — and hence all $\hat{\boldsymbol{T}}_i^{(0)}$ — are committed separately from the other $\hat{\boldsymbol{T}}$ matrices (they are in $\mathsf{cm}_{\boldsymbol{X}}$). Consequently, when $\mathcal{P}$ and $\mathcal{V}$ execute Algorithm 6, the values of $\widetilde{\boldsymbol{M}}$ are not contained in a single square-root commitment, and it is no longer immediate how to perform the evaluation, since both square-root commitments may have different shapes. We address this issue with the witness-injection technique of [20, Section 6.2][1].

In addition to combining multiple commitments, we must also account for public values appearing in the evaluation. Following [20] and subsequent work, $\mathcal{V}$ locally evaluates the public contribution at the corresponding random point, obtaining a value ζ, and subtracts this from $t_{\boldsymbol{M}}$ by multiplying its commitment with $g^{-\zeta}$ before invoking Algorithm 6.

Remark 5.2 (Fiat–Shamir Instantiation)**.** *The protocol described in Algorithm 5 is an* interactive *proof system for the local training relation $\mathcal{R}_{\mathsf{train}}$. In* FALAFEL*, this interactive protocol is made non-interactive by applying the Fiat–Shamir transform, yielding the proof $\pi_{k,r}$ that is broadcast during training (cf. Step 12 of Algorithm 4).*

Theorem 5.3. FALAFEL*'s client side protocol, defined by applying the Fiat–Shamir transform to Algorithm 5, is complete, knowledge sound and zero-knowledge.*

Proof. As shown in Appendix E, all the intermediate protocols are complete, $\boldsymbol{k}$-out-of-$\boldsymbol{N}$ special sound (for specified $\boldsymbol{k}$ and $\boldsymbol{N}$) and honest verifier zero-knowledge. After applying the Fiat–Shamir transformation to each intermediate protocol, we obtain a non-interactive commit-and-prove ZKP for the same relations, that is complete, knowledge sound and zero knowledge. Because of [25], we know that knowledge soundness holds under the same conditions as their interactive counterparts. Thus, Algorithm 5 is a composition of non-interactive ZKPs following the commit-and-prove paradigm; the result follows from the proof of [4, Theorem 3.1]. □

1. Specifically, we follow the approach outlined in [20, footnote 6], with a minor optimization tailored to our setting: due to the structured layout of the injected witness values, the injected portion can itself be evaluated via Algorithm 6, using only the relevant subset of $\boldsymbol{L}$ and $\boldsymbol{R}$ of step 1.

5.4. Training Phase — Server-Side

The server-side protocol for proving the correctness of training in round r corresponds to steps 18 and 19 of Algorithm 4. In round r, S aggregates the weights received from each client P_k according to $\boldsymbol{\mathcal{W}}_{\text{glob}} \coloneqq \frac{1}{K}\sum_k \boldsymbol{\mathcal{W}}_k$ and sends these new global weights to all clients. However, at this point the protocol is not finished: each client must be able to verify that S performed the aggregation correctly, based on the commitments to the clients' local weights.

Since aggregation is linear and the commitment scheme is homomorphic, the server can aggregate the commitments analogously. After reconstructing the commitments to the P_k's final weights $\boldsymbol{\mathcal{W}}_k$ by using P_k's commitment to $\boldsymbol{\mathcal{W}}_0$ and $\overline{\boldsymbol{G}}_i^{(\ell)}$ (cf. Section 5.3), let $\mathsf{cm}_{j,k}$ denote the j-th vector commitment to $\boldsymbol{\mathcal{W}}_k$, with associated randomness $r_{j,k}$. For each j one can compute $\mathsf{cm}_j \coloneqq \prod_k (\mathsf{cm}_{j,k})^{1/K}$, which is the j-th commitment in our square-root commitment to $\boldsymbol{\mathcal{W}}_{\text{glob}}$.

The clients want to verify that these aggregated commitments correspond to the global weights received from S. By construction, the randomness associated with cm_j equals $\frac{1}{K}\sum_k r_{j,k}$. While S has access to this randomness, it cannot simply reveal it, as doing so would violate the threat model (cf. Section 3.2). In particular, $K-1$ colluding clients could use it to recover the randomness of the remaining client and open its original commitment. To avoid this, S re-randomizes the aggregated commitments by multiplying them with zero-commitments, obtaining $\overline{\mathsf{cm}}_j$. These are broadcast, and the corresponding fresh randomness is sent to the clients, such that they can use it to construct their proof in the following round. Finally, S proves that cm_j and $\overline{\mathsf{cm}}_j$ indeed commit to the same values. This is efficiently achieved using a standard (batched) proof-of-equality between commitments.

Security. The server-side aggregation protocol is relatively straightforward. It consists of aggregation and re-randomization of commitments, and a standard (batched) proof-of-equality. As a result, it is complete, knowledge-sound, and honest-verifier perfect zero-knowledge.

5.5. Usage Phase

Any verifier can validate the training process only requiring the final aggregated weights $\boldsymbol{\mathcal{W}}_{\text{final}}$ along with a transcript tr containing: (1) auditor signatures $\{\sigma_{\mathsf{cm}_{D_k}}\}_{k\in[K]}$; (2) client proofs $\{\pi_{k,r}\}_{k\in[K],r\in[N_{\text{rounds}}]}$; and (3) aggregation proofs $\{\pi_{\text{glob},r}\}_{r\in[N_{\text{rounds}}]}$. Using pp, NN, and tr, $\mathcal{V}$ confirms that the model was trained correctly on authenticated data.

TABLE 1. EXPERIMENTAL CONFIGURATIONS FOR TABLE 2; EACH VARIANT MODIFIES ONE PARAMETER COMPARED TO `base`.

Config	NN	N_{samples}	S_{batch}	Iters/epoch
`base`	8−3×8−8	32	16	2
`data`	8−3×8−8	128	64	2
`iters`	8−3×8−8	32	2	16
`wide`	32−3×32−32	32	16	2
`deep`	8−8×8−8	32	16	2

6. Implementation and Evaluation

We have implemented both the client and server in Rust 1.94.1, using the *arkworks* framework [26]. Our source code is publicly available at https://github.com/xQiratNL/FALAFEL. We leverage parallelization across available CPU cores for multi-scalar multiplications as well as other computationally expensive steps, which better reflects scalability in practical deployments. We note that additional parallelization opportunities likely remain. Moreover, our proof-of-scaling (or activation) is implemented using the verifier optimization techniques described in [13, Section 6.2]. We benchmark our implementation on fully connected ReLU layers, using integers in the 60–64 bit range to represent model parameters, depending on the model configuration. In line with Section 2, an *iteration* corresponds to a single training step on one mini-batch, while an *epoch* denotes a full pass over the entire local dataset, i.e., N_{batch} iterations.

All experiments below are run on a 2.2 GHz Intel Xeon Gold machine with 128 cores and 128 GB of RAM. The security level λ depends on the binding property of the Pedersen commitment, and the ZKP's soundness error. By [25] the latter is $(Q+1)\kappa$, where Q as the allowed query complexity of the adversary and $\kappa = 1 - \prod_i (1 - (k_i - 1)/N_i)$ for any $\boldsymbol{k}$-out-of-$\boldsymbol{N}$ special sound protocol. Following Theorem 5.3 and the theorems in Appendix E, we require that Q and κ are negligible in 2^λ, i.e., for κ we require $q \cdot E \cdot N_{\text{samples}} \cdot \sum_i n_i$ to be negligible in 2^λ. We instantiate the group used for the Pedersen commitment scheme (and thus also our ZKPs) using *Curve25519* [27], which results in ∼128-bit security. The random oracle is instantiated using *Keccak-f1600* [28].

6.1. Performance

Performance Breakdown. We evaluate how different configurations affect the runtime of the various phases of FALAFEL; the results are shown in Table 2. Each phase corresponds to the client's steps with the same name in Algorithm 5, where *Commitments* refers to the generation of $\mathsf{cm}_{\text{trace}}$ and *Server Aggregation* to the aggregation proof by the server. The `base` configuration uses fully connected ReLU layers with 8 input neurons, three hidden layers of width 8, and 8 output neurons. Each client trains on 32 local samples drawn from its local dataset, with a batch size of 16, resulting in two iterations per epoch. Each additional configuration modifies a single parameter, as shown in Table 1.

In Table 2 we break down the prover cost of a single client, and show the cost of the aggregation done by the server. Across all configurations, most time is spent on the forward pass, backward pass, and weight update, this is caused by the comparatively high cost of the proof-of-scaling (or activation). In the `base` configuration, the matrices for the proof-of-scaling (or activation) have roughly the same size, making the runtime similar. When the dataset size increases, the forward and backward passes become slower, while the weight update remains largely unchanged, since the number of values for the proof-of-scaling only increases for the first two. For wider networks, all phases become slower, with the weight update dominating, again due to the proof-of-scaling. Deeper networks do not really affect the distribution of the costs, which is expected, since all phases depend linearly on the number of layers. Increasing the number of iterations does shift the dominant cost to the weight update. This is mainly due to having to compute a commitment to the final weights from more commitments to $\overline{\boldsymbol{G}}_i^{(\ell)}$.

Across all configurations, proof sizes remain fairly stable. Finally, since server-side aggregation is much faster than the client-side proofs, the overhead introduced by federation is small. Moreover, we observe a good balance between prover and verifier time, meaning that we did not sacrifice prover cost to improve verifier cost.

Scaling — NN Width. Second, we examine how FALAFEL

TABLE 2. BREAKDOWN BY PHASE AND CONFIGURATION.

Phase	`base`	`data`	`iters`	`wide`	`deep`
Commitments	0.05 s	0.07 s	0.10 s	0.12 s	0.06 s
Batch correctness	0.10 s	0.14 s	0.10 s	0.14 s	0.10 s
Forward pass	2.73 s	8.96 s	2.75 s	9.51 s	9.01 s
Backward pass	3.15 s	9.87 s	3.34 s	10.46 s	9.49 s
Weight update	2.52 s	2.52 s	17.92 s	36.68 s	8.94 s
Poly. evals	0.49 s	0.60 s	0.68 s	0.68 s	0.53 s
Server Aggregation	0.17 s	0.27 s	0.40 s	0.38 s	0.21 s
Total Prover	**9.22 s**	**22.43 s**	**25.30 s**	**57.98 s**	**28.34 s**
Total Verifier	**2.47 s**	**6.44 s**	**6.84 s**	**16.28 s**	**8.13 s**
Total Proof Size	**49.6 kB**	**65.8 kB**	**80.8 kB**	**78.9 kB**	**56.3 kB**

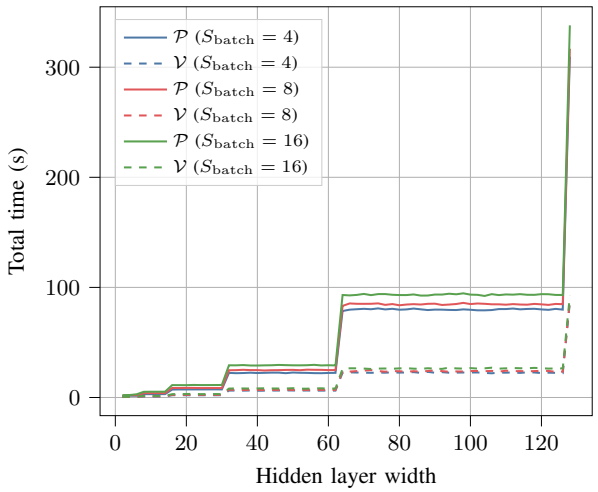

Figure 2. $\mathcal{P}$ and $\mathcal{V}$ total runtime for varying hidden layer widths.

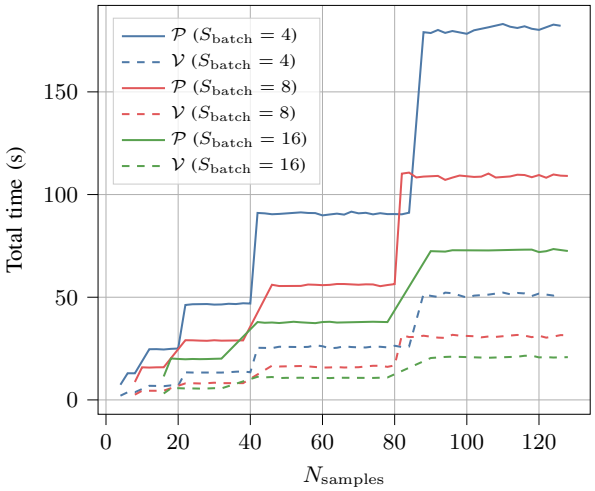

Figure 3. Prover and verifier runtime for various dataset sizes.

scales on a fully connected ReLU network with three hidden layers, where each layer has between 2 and 128 neurons. Having one or two layers be smaller, would result in the same runtime, due to how the square-root commitments and sumchecks are structured, i.e., we care only about the maximum layer width. The corresponding runtimes are shown in Figure 2. We observe noticeable jumps in both runtime and proof size when the network width reaches powers of two. This behavior is expected, as the weight matrices are padded such that the dimension equals the next greater 2-power, which is required for the sumcheck proofs. Since the size of the weight matrices largely determines the cost of the weight-update phase, this phase dominates the overall runtime of the protocol. In contrast, the batch size has little effect on the runtime and only a small effect on the proof size in our experiments, as can be seen in Figure 2. This is due to the batch size being much smaller than the network width, making the increase of cost for the forward and backward pass small compared to the increase in cost for the weight update. The proof size shows jumping behavior analogous to the runtime, ranging from an initial $\sim$30 kB, to slightly below $\sim$100 kB for a layer weight of 128 and up.

Scaling — Dataset Size. Finally, we study how FALAFEL scales with increasing dataset sizes. Figure 3 shows the total runtime for varying dataset sizes and batch sizes. In contrast to the previous experiments, we clearly observe differences between batch sizes in runtime. For the proof size we observe identical behavior, with proof sizes ranging from 50 kB to 200 kB for $S_{\text{batch}} = 4$ and 100 kB for $S_{\text{batch}} = 16$.

In general, larger batch sizes result in fewer iterations per epoch, which leads to significantly more efficient proving and verification times, as well as smaller proof sizes. This is in contrast to many prior schemes, e.g., Kaizen [3], and benefits practical performance of FALAFEL on realistic batch sizes. Additionally, we observe that Kaizen reports runtimes on a per-iteration basis due to stronger amortization effects. While FALAFEL also benefits from amortization, this effect is less pronounced in our setting.

TABLE 3. COMPARISON OF $\mathcal{P}$ RUNTIME PER ITERATION, TOTAL $\mathcal{V}$ RUNTIME (IN SECONDS), AND CLIENT PROOF SIZE WITH PRIOR WORK.

	S_{batch}	Nova	Kaizen	FALAFEL
Prover (s)	4	–	112	**145**
	8	–	142	**155**
	16	1868	193	**173**
Verifier (s)	4	–	0.062	**42**
	8	–	0.065	**49**
	16	1677	0.073	**55**
Proof Size (kB)	4	–	920	**60.7**
	8	–	1009	**63.1**
	16	9.80	1021	**74.9**

6.2. Comparison

Finally, we evaluate the performance of FALAFEL on large networks comparable in size to LeNet, with $\sim$60,000 trainable parameters. While LeNet itself includes convolutional and pooling layers, our benchmarks focus on fully connected layers, which are slightly more expensive to compute and to prove. We compare against a generic construction based on *Halo* [29], representing the state of the art in incremental verifiable computation, and the dedicated zkPoT scheme *Kaizen* [3] as it is the work closest to ours and shows better performance than alternative prior work, on a similar security level (our 128 bits versus their 100 bits), with the important difference that they instantiate the random oracle using a less-vetted arithmetic hash function.

We report the prover runtime per iteration, the total verifier runtime, and the proof size in Table 3. The results for Halo and Kaizen are taken directly from [3]. We note that Kaizen was evaluated on hardware with twice as many CPU cores as ours. In addition, FALAFEL supports substantial parallelization, further reducing runtimes in practice. We could not determine Kaizen's degree of parallelization. Table 3 should thus be interpreted as indicative of relative performance rather than as a strictly fair head-to-head comparison.

The main difference between our construction and Kaizen lies in how proofs are aggregated across iterations. Kaizen introduces a custom polynomial aggregation step that combines all per-iteration proofs into a single aggregate proof, allowing the verifier to perform only a lightweight check. This aggregation step is the dominant cost in Kaizen, but it is executed only once and therefore amortizes well over many iterations. As a result, although our prover runtime per iteration is comparable to or faster than Kaizen, the total verification time of the full zk-PoT is significantly lower for Kaizen. On the other hand, our proof sizes are substantially smaller, and our comparable prover time does not rely on heavy amortization over a large number of rounds.

Overall, FALAFEL seems better suited to settings with a small number of iterations per epoch and low communication overhead, both of which are typical in FL scenarios.

7. Conclusion & Discussion

We presented a ZKP-based scheme for publicly verifiable FL with active security on authenticated data. FALAFEL achieves prover times comparable to existing work and significantly smaller proofs, while relying exclusively on standard cryptographic assumptions. Its modular design further enables future extensions and improvements by replacing individual core components with alternative constructions.

A limitation of our work is the relatively high verification time compared to some existing schemes. We believe it difficult to substantially reduce this cost without adopting different underlying assumptions. In the FL setting targeted by FALAFEL, we consider this overhead acceptable, particularly since clients typically spend more time on proof generation. From this perspective, the more relevant observation is that the prover runtime and proof size are practical and either comparable to or better than those of alternative solutions.

Finally, FALAFEL can be extended to support convolutional layers, for example following the approach of [3]. In addition, adaptations to other piecewise-linear activation functions, or suitable approximations thereof, should be feasible within our framework. We chose not to include these extensions in this work in order to focus on our core contributions, and because they would neither substantially affect the security guarantees nor change the performance trends demonstrated in our evaluation.

Acknowledgements

This work was partially funded by the European Union's Horizon Europe programme (Grant Agreement No. 101217315) and supported by TNO's Early Research Program *Next Generation Cryptography*.

References

[1] M. Maruseac, S. Meiklejohn, M. Lodato, and Google Open Source Security Team (GOSST), "Increasing transparency in AI security," Oct. 2023. [Online]. Available: https://security.googleblog.com/2023/10/increasing-transparency-in-ai-security.html

[2] K. Balan, R. Learney, and T. Wood, "A framework for cryptographic verifiability of end-to-end AI pipelines," in *IWSPA@CODASPY*. ACM, 2025, pp. 49–59.

[3] K. Abbaszadeh, C. Pappas, D. Papadopoulos, and J. Katz, "Zero-Knowledge Proofs of Training for Deep Neural Networks," 2024. [Online]. Available: https://eprint.iacr.org/2024/162

[4] M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs," 2019. [Online]. Available: https://eprint.iacr.org/2019/142

[5] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: lightweight sublinear arguments without a trusted setup," *Des. Codes Cryptogr.*, vol. 91, no. 11, pp. 3379–3424, 2023.

[6] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. P. Ward, "Marlin: Preprocessing zksnarks with universal and updatable SRS," in *EUROCRYPT (1)*, ser. Lecture Notes in Computer Science, vol. 12105. Springer, 2020, pp. 738–768.

[7] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *CRYPTO (3)*, ser. Lecture Notes in Computer Science, vol. 11694. Springer, 2019, pp. 733–764.

[8] H. Sun and H. Zhang, "zkdl: Efficient zero-knowledge proofs of deep learning training," *CoRR*, vol. abs/2307.16273, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.16273

[9] S. Garg, A. Goel, S. Jha, S. Mahloujifar, M. Mahmoody, G. Policharla, and M. Wang, "Experimenting with zero-knowledge proofs of training," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 1880–1894. [Online]. Available: https://doi.org/10.1145/3576915.3623202

[10] S. Waiwitlikhit, I. Stoica, Y. Sun, T. Hashimoto, and D. Kang, "Trustless audits without revealing data or models," in *ICML*. OpenReview.net, 2024.

[11] H. Duan, Z. Peng, L. Xiang, Y. Hu, and B. Li, "A Verifiable and Privacy-Preserving Federated Learning Training Framework," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 5, pp. 5046–5058, Sep. 2024.

[12] G. Keshavarzkalhori, C. Pérez-Solà, G. Navarro-Arribas, J. Herrera-Joancomartí, and H. Yajam, "Federify: A verifiable federated learning scheme based on zksnarks and blockchain," *IEEE Access*, vol. 12, pp. 3240–3255, 2024.

[13] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 315–334.

[14] D. Khovratovich, R. D. Rothblum, and L. Soukhanov, "How to prove false statements: Practical attacks on fiat-shamir," *IACR Cryptol. ePrint Arch.*, p. 118, 2025.

[15] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated Optimization: Distributed Machine Learning for On-Device Intelligence," Oct. 2016.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[17] P. Liu, X. Xu, and W. Wang, "Threats, attacks and defenses to federated learning: Issues, taxonomy and perspectives," *Cybersecurity*, vol. 5, no. 1, p. 4, Feb. 2022.

[18] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *Advances in Cryptology — CRYPTO '91*, ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer, 1992, pp. 129–140.

[19] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," in *FOCS*. IEEE Computer Society, 1990, pp. 2–10.

[20] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-Efficient zkSNARKs Without Trusted Setup," in *2018 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2018, pp. 926–943.

[21] C. P. Schnorr, "Efficient Identification and Signatures for Smart Cards," in *Advances in Cryptology — CRYPTO' 89 Proceedings*, G. Brassard, Ed. New York, NY: Springer, 1990, pp. 239–252.

[22] J. Thaler, "Time-Optimal Interactive Proofs for Circuit Evaluation," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 8043, pp. 71–89.

[23] J. Groth, "Linear algebra with sub-linear zero-knowledge arguments," in *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '09. Berlin, Heidelberg: Springer-Verlag, 2009, p. 192–208. [Online]. Available: https://doi.org/10.1007/978-3-642-03356-8_12

[24] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish, "A Hybrid Architecture for Interactive Verifiable Computation," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, ser. SP '13. USA: IEEE Computer Society, May 2013, pp. 223–237.

[25] T. Attema, S. Fehr, M. Klooß, and N. Resch, "The Fiat—Shamir Transformation of $(\Gamma_1, \ldots, \Gamma_\mu)$-Special-Sound Interactive Proofs," *Journal of Cryptology*, vol. 39, no. 2, p. 19, Mar. 2026.

[26] arkworks contributors, "arkworks: A zksnark ecosystem," https://arkworks.rs, 2022.

[27] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Heidelberg: Springer, 2006, pp. 207–228.

[28] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," U.S. Department of Commerce, Tech. Rep. Federal Information Processing Standard (FIPS) 202, Aug. 2015.

[29] A. Kothapalli, S. T. V. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *CRYPTO (4)*, ser. Lecture Notes in Computer Science. Springer, 2022, pp. 359–388.

[30] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," in *USENIX Security Symposium*. USENIX Association, 2021, pp. 501–518.

[31] T. Liu, X. Xie, and Y. Zhang, "zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy," 2021. [Online]. Available: https://eprint.iacr.org/2021/673

[32] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *NIPS*, 2017, pp. 4672–4681.

[33] H. Sun, J. Li, and H. Zhang, "zkllm: Zero knowledge proofs for large language models," in *CCS*. ACM, 2024, pp. 4405–4419.

[34] D. Balbás, D. Fiore, M. I. G. Vasco, D. Robissout, and C. Soriente, "Modular sumcheck proofs with applications to machine learning and image processing," in *CCS*. ACM, 2023, pp. 1437–1451.

[35] S. Lee, H. Ko, J. Kim, and H. Oh, "vcnn: Verifiable convolutional neural network based on zk-snarks," *IEEE Trans. Dependable Secur. Comput.*, vol. 21, no. 4, pp. 4254–4270, 2024.

[36] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 911–926, 2020.

[37] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, "Veriml: Enabling integrity assurances and fair payments for machine learning as a service," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 10, pp. 2524–2540, 2021.

[38] H. Jia, M. Yaghini, C. A. Choquette-Choo, N. Dullerud, A. Thudi, V. Chandrasekaran, and N. Papernot, "Proof-of-learning: Definitions and practice," in *SP*. IEEE, 2021, pp. 1039–1056.

[39] T. Attema, "Compressed Σ-protocol theory," PhD thesis, Leiden University, 2023.

[40] T. Attema, S. Fehr, and N. Resch, "Generalized special-sound interactive proofs and their knowledge soundness," in *TCC (3)*, ser. Lecture Notes in Computer Science, vol. 14371. Springer, 2023, pp. 424–454.

Appendix A. Additional Related work

Besides schemes focusing on training verification and verification of FL, there is a multitude of works presenting schemes for verifiable inference, e.g., [30], [31], [32], [33], [34], [35]. Some of the techniques introduced in these works are also useful for zk-PoTs; however, in isolation they are insufficient to obtain efficient zk-PoT constructions. Moreover, these approaches do not address the training phase of the full pipeline and therefore do not provide a suitable basis for our work. Nevertheless, we observe that sumcheck-based techniques generally lead to practical efficiency.

Other schemes related to FALAFEL include *VerifyNet* [36], which employs double-masking protocols to verify server-side aggregation results, and *VeriML* [37], which provides verifiable gradient aggregation using secret-sharing schemes. Neither approach offers zero-knowledge guarantees, and *VeriML* provides only probabilistic correctness guarantees. Finally, Jia et al. [38] introduce the idea of logging training checkpoints to provide evidence of computational expenditure; however, privacy guarantees are not considered. We do not see a clear way to extend these works to our setting while simultaneously achieving zero-knowledge and practical efficiency.

Appendix B. Additional Preliminaries

For *completeness* we follow the standard definition and therefore do not reiterate it. Given the different flavors in *knowledge soundness* and *zero-knowledge* definitions, we present the definitions we rely on below. For knowledge soundness we rely on the notion of $\boldsymbol{k}$-out-of-$\boldsymbol{N}$ special-soundness [39, Def. 2.34]: a $\boldsymbol{k}$-out-of-$\boldsymbol{N}$ special-sound protocol is knowledge sound.

Definition B.1 (Tree of Transcripts)**.** *Let $\boldsymbol{k} = (k_1, \ldots, k_\mu) \in \mathbb{N}^\mu$. A $\boldsymbol{k}$-tree of transcripts for a $(2\mu+1)$-round public-coin interactive proof Π is a set of $K = \prod_{i=1}^{\mu} k_i$ transcripts arranged in the following tree structure. The nodes in this tree correspond to the prover's messages and the edges to the verifier's challenges. Every node at depth i has precisely k_i children corresponding to k_i pairwise distinct challenges. Every transcript corresponds to exactly one path from the root node to a leaf node. We refer to the corresponding tree of challenges as a $\boldsymbol{k}$-tree of challenges.*

Definition B.2 ($\boldsymbol{k}$-out-of-$\boldsymbol{N}$ Special-Soundness)**.** *Let $\boldsymbol{k} = (k_1, \ldots, k_\mu)$, $\boldsymbol{N} = (N_1, \ldots, N_\mu) \in \mathbb{N}^\mu$. A $(2\mu+1)$-round public-coin interactive proof Π for relation $\mathcal{R}$, where $\mathcal{V}$ samples the i-th challenge from a set of cardinality $N_i \geq k_i$ for $1 \leq i \leq \mu$, is $\boldsymbol{k}$-out-of-$\boldsymbol{N}$ special-sound if there exists a polynomial-time algorithm that, on input a statement x and a $\boldsymbol{k}$-tree of accepting transcripts, outputs a witness $w \in \mathcal{R}(x)$. We also say Π is $\boldsymbol{k}$-special-sound.*

As our protocols are used in the Fiat–Shamir model, it suffices to establish honest-verifier zero-knowledge rather than full zero-knowledge.

Definition B.3 ((Special) Honest-Verifier Zero-Knowledge)**.** *An interactive proof $\Pi = (\mathcal{P}, \mathcal{V})$ for relation $\mathcal{R}$ is said to be (statistical) **honest-verifier zero-knowledge** (HVZK) if there exists a polynomial time simulator $\mathcal{S}$ such that the following families of random variables are statistically indistinguishable:*

- $\{\textbf{\textit{view}}^{\mathcal{V}}_{\mathcal{P}}(x;w) : (x;w) \in \mathcal{R}\}$, *where* $\textbf{\textit{view}}^{\mathcal{V}}_{\mathcal{P}}(x;w)$ *describes* $\mathcal{P}$*'s messages and* $\mathcal{V}$*'s random tape when evaluating* $\Pi = (\mathcal{P}, \mathcal{V})$ *on input* $(x;w)$*;*
- $\{\mathcal{S}(x) : (x;w) \in \mathcal{R}\}$.

If Δ *denotes the statistical distance and* $\Delta\big(\textbf{\textit{view}}^{\mathcal{V}}_{\mathcal{P}}(x;w), \mathcal{S}(x)\big) = 0$ *for all* $(x;w) \in \mathcal{R}$, Π *is said to be perfectly HVZK. If these families of random variables are only computationally indistinguishable,* Π *is said to be computationally HVZK. Further, if the simulator proceeds by first sampling the verifier's messages uniformly at random,* Π *is said to be special honest-verifier zero-knowledge (SHVZK).*

Appendix C. Supporting Protocols

C.1. Proof of Dot Product

FALAFEL uses Hyrax's proof$_{\mathsf{log}}$-of-dot-prod [20, Figs. 7–8], which adapts Bulletproof's inner-product argument to the setting where one vector is committed to and the other is public. However, the soundness analysis of Hyrax is somewhat outdated. Using more modern terminology and techniques introduced after its publication, we observe that the Hyrax protocol is in fact $(3, \ldots, 3, 2)$-special sound, as it corresponds to a compressed Σ-protocol [39, Protocol 3]. Note that this means that $2 \cdot 3^{\log n} = 2n^{\log(3)}$ transcripts are required to extract, rather than the $n + 2$ claimed by Hyrax.

C.2. Proof of Polynomial Evaluation

We restate the polynomial evaluation from Hyrax. Throughout we use the notation introduced in Section 2. The protocol allows a prover to convince a verifier that cm_t is a commitment to the evaluation of the multilinear extension of a private matrix $\boldsymbol{Y} \in \mathbb{F}^{2^\ell \times 2^\ell}$ at a public point $(\boldsymbol{r}_1, \boldsymbol{r}_2)$. In what follows, $\boldsymbol{y}_k$ denotes that k-th row of the matrix $\boldsymbol{Y}$, using 0-indexing.

Definition C.1 (Polynomial Evaluation Relation)**.** *The polynomial evaluation relation* $\mathcal{R}_{\mathsf{poly\text{-}eval}}$ *is defined over public inputs* x *and witnesses* w*. We have* $(x;w) \in \mathcal{R}_{\mathsf{poly\text{-}eval}}$ *if and only if*

$$x = \left(\{\mathsf{cm}_{\boldsymbol{y}_k}\}_{k=0}^{2^\ell-1}, \mathsf{cm}_t, \boldsymbol{r}_1, \boldsymbol{r}_2\right),$$

$$w = \left(\boldsymbol{Y}, t, \{\rho_k\}_{k=0}^{2^\ell-1}, \rho_t\right),$$

such that $t = \widetilde{\boldsymbol{Y}}(\boldsymbol{r}_1, \boldsymbol{r}_2), \mathsf{cm}_t = \mathsf{Com}(t, \rho_t)$*, and for all* $k = 0, \ldots, 2^\ell - 1$*:* $\mathsf{cm}_{\boldsymbol{y}_k} = \mathsf{Com}(\boldsymbol{y}_k, \rho_k)$.

To formulate a zero-knowledge proof for the relation $\mathcal{R}_{\mathsf{poly\text{-}eval}}$ we introduce some notation. Let $\boldsymbol{s} \coloneqq \boldsymbol{r}_1 \,\|\, \boldsymbol{r}_2 \in \mathbb{F}^{2\ell}$ be the concatenation of $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$. Then

$$\widetilde{\boldsymbol{Y}}(\boldsymbol{r}_1, \boldsymbol{r}_2) = \sum_{\boldsymbol{b} \in \{0,1\}^{2\ell}} \boldsymbol{Y}_{\boldsymbol{b}}\, \chi_{\boldsymbol{b}}(\boldsymbol{s}),$$

Algorithm 6 Zero-Knowledge Proof for $\mathcal{R}_{\mathsf{poly\text{-}eval}}$

Public Input: $x = \left(\{\mathsf{cm}_{\boldsymbol{y}_k}\}_{k=0}^{2^\ell-1}, \mathsf{cm}_t, \boldsymbol{r}_1, \boldsymbol{r}_2\right)$

Witness: $w = \left(\boldsymbol{Y}, t, \{\rho_k\}_{k=0}^{2^\ell-1}, \rho_t\right)$

1: $\mathcal{P}$ and $\mathcal{V}$ compute vectors $\boldsymbol{L}$ and $\boldsymbol{R}$ from the public evaluation point $(\boldsymbol{r}_1, \boldsymbol{r}_2)$ as in (C.1).
2: $\mathcal{V}$ computes the linear combination of row commitments $\mathsf{cm}_{\boldsymbol{t}'} \coloneqq \prod_{k=0}^{2^\ell-1} (\mathsf{cm}_{\boldsymbol{y}_k})^{\boldsymbol{L}_{k+1}}$.
3: $\mathcal{P}$ and $\mathcal{V}$ execute proof$_{\log}$-of-dot-product on $(\mathsf{cm}_{\boldsymbol{t}'}, \boldsymbol{R}, \mathsf{cm}_t)$.
4: $\mathcal{V}$ accepts if and only if the proof$_{\log}$-of-dot-product verifies.

where

$$\chi_{\boldsymbol{b}}(\boldsymbol{s}) \coloneqq \prod_{k=1}^{2\ell} \chi_{b_k}(s_k), \qquad \chi_{b_k}(s_k) \coloneqq s_k b_k + (1-s_k)(1-b_k),$$

and $\boldsymbol{Y}_{\boldsymbol{b}}$ denotes the entry of $\boldsymbol{Y}$ indexed by the binary vector $\boldsymbol{b} \in \{0,1\}^{2\ell}$. We split $\chi_{\boldsymbol{b}}(\boldsymbol{s})$ into row and column components:

$$\check{\chi}_{\boldsymbol{b}} \coloneqq \prod_{k=1}^{\ell} \chi_{b_k}(s_k), \qquad \hat{\chi}_{\boldsymbol{b}} \coloneqq \prod_{k=\ell+1}^{2\ell} \chi_{b_k}(s_k).$$

Next, define the vectors of evaluated basis polynomials

$$\begin{aligned} \boldsymbol{L} &\coloneqq (\check{\chi}_{\boldsymbol{0}}, \check{\chi}_{\boldsymbol{1}}, \ldots, \check{\chi}_{\boldsymbol{2^\ell-1}}) \in \mathbb{F}^{2^\ell}, \\ \boldsymbol{R} &\coloneqq (\hat{\chi}_{\boldsymbol{0}}, \hat{\chi}_{\boldsymbol{2^\ell}}, \ldots, \hat{\chi}_{\boldsymbol{2^\ell \cdot (2^\ell-1)}}) \in \mathbb{F}^{2^\ell}, \end{aligned} \tag{C.1}$$

where the indices are integer representations of the underlying bitstrings. We present a zero-knowledge proof for $\mathcal{R}_{\mathsf{poly\text{-}eval}}$ in Algorithm 6. It's security is proven in Appendix E.

C.3. Aggregated Proof of Sumcheck

We restate the zero-knowledge protocol for aggregated sumchecks from Hyrax in Algorithm 7, recalling the required machinery below. We make minor adaptations, revolving around the fact that we only need commitments to three coefficients (rather than four) due to our matrix structure. Additionally, we introduce notational changes and adapt the explanation to our setting.

The sumcheck protocol will be executed on the function $g(\boldsymbol{x})$ which satisfies $\sum_{\boldsymbol{r}_3 \in \{0,1\}^\ell} g(\boldsymbol{r}_3) = \sum_{\boldsymbol{r}_3 \in \{0,1\}^\ell} \widetilde{\boldsymbol{A}}(\boldsymbol{r}_1, \boldsymbol{r}_3) \cdot \widetilde{\boldsymbol{B}}(\boldsymbol{r}_3, \boldsymbol{r}_2)$ for given $\boldsymbol{r}_1, \boldsymbol{r}_2$, verifying that it equals $\widetilde{\boldsymbol{C}}(\boldsymbol{r}_1, \boldsymbol{r}_2)$. Observe that g has degree at most 2 in every variable. Therefore, the univariate polynomial g_j in round j also has degree at most 2, meaning that it can be described by (commitments to) three coefficients, $c_{2,j}, c_{1,j}, c_{0,j}$. Thus, $\mathcal{V}$ checks that $g_{j-1}(r_{3,j-1}) = g_j(0) + g_j(1)$, i.e., $c_{2,j-1} r_{3,j-1}^2 + c_{1,j-1} r_{3,j-1} + c_{0,j-1} = c_{2,j} + c_{1,j} + 2c_{0,j}$. Analogously, in round $j = 1$, $\mathcal{V}$ checks that $t_{\boldsymbol{C}} = g_1(0) + g_1(1)$. After the final round $\mathcal{V}$ also needs to check whether $g_\ell(r_\ell) = t_{\boldsymbol{A}} \cdot t_{\boldsymbol{B}}$, i.e., whether $c_{2,\ell} r_\ell^2 + c_{1,\ell} r_\ell + c_{0,\ell} = t_{\boldsymbol{AB}}$. Thus, we can express all $\mathcal{V}$'s checks linearly in terms of the

Algorithm 7 Zero-Knowledge Proof for $\mathcal{R}_{\text{agg-sum}}$

Public Input: $x = \left(\{\mathsf{cm}_{\boldsymbol{c}_j}\}, \mathsf{cm}_{t_{\boldsymbol{C}}}, \mathsf{cm}_{t_{\boldsymbol{AB}}}, \boldsymbol{r}_3\right)$
Witness: $w = \left(\{\boldsymbol{c_j}\}_{j\in[\ell]}, \{r_{\boldsymbol{c}_j}\}_{j\in[\ell]}, t_{\boldsymbol{C}}, r_{\boldsymbol{C}}, t_{\boldsymbol{AB}}, r_{\boldsymbol{AB}}\right)$

1: $\mathcal{P}$ computes $\boldsymbol{\pi}$ as in (C.2).
2: $\mathcal{P}$ and $\mathcal{V}$ compute $\boldsymbol{M}$ as in (C.3).
3: $\mathcal{P}$ randomly samples $\boldsymbol{d} := (d_{2,1}, d_{1,1}, d_{0,1}, d_{2,2}, \ldots, d_{2,\ell}, d_{1,\ell}, d_{0,\ell}) \in \mathbb{F}^{3\ell}$. $\mathcal{P}$ now calculates the commitment $\mathsf{cm}_{\boldsymbol{d}_j}$ for each $j \in [\ell]$, where $\boldsymbol{d}_j := (d_{2,j}, d_{1,j}, d_{0,j})$, using randomness $r_{\boldsymbol{d}_1}, \ldots, r_{\boldsymbol{d}_\ell} \in \mathbb{F}$. $\mathcal{P}$ shares the commitments $\mathsf{cm}_{\boldsymbol{d}_j}$ with $\mathcal{V}$ for $j \in [\ell]$.
4: $\mathcal{V}$ randomly samples $\rho_1, \ldots, \rho_{\ell+1} \in \mathbb{F}$ and sends them to $\mathcal{P}$.
5: $\mathcal{P}$ randomly samples $r_P \in \mathbb{F}$ and sends cm_P to $\mathcal{V}$, where cm_P is the commitment to $P := \langle \sum_{k=1}^{\ell+1} \rho_k \cdot \hat{\boldsymbol{m}}_k, \boldsymbol{d}\rangle$ with randomness r_P.
6: $\mathcal{V}$ sends a random challenge $c \in \mathbb{F}$ to $\mathcal{P}$.
7: $\mathcal{P}$ computes and sends the following to $\mathcal{V}$:

$$\begin{aligned} \boldsymbol{z} &= c \cdot \hat{\boldsymbol{\pi}} + \boldsymbol{d}, \\ r_{\boldsymbol{z}_j} &= c \cdot r_{\boldsymbol{c}_j} + r_{\boldsymbol{d}_j} \text{ for all } j \in [\ell], \text{ and} \\ r &= c \cdot (\rho_1 r_{\boldsymbol{C}} + \rho_{\ell+1} r_{\boldsymbol{AB}}) + r_P. \end{aligned}$$

8: $\mathcal{V}$ receives $\boldsymbol{z} = (z_{2,1}, z_{1,1}, z_{0,1}, \ldots, z_{2,\ell}, z_{1,\ell}, z_{0,\ell})$ and does the following for each $j \in [\ell]$:
- Compute $\mathsf{cm}_{\boldsymbol{z}_j}$, where $\boldsymbol{z}_j = (z_{2,j}, z_{1,j}, z_{0,j})$, using randomness $r_{\boldsymbol{z}_j}$.
- Check whether $\mathsf{cm}_{\boldsymbol{z}_j}$ equals $(\mathsf{cm}_{c_j})^c \cdot \mathsf{cm}_{d_j}$

9: Check whether the commitment to $\langle \sum_{k=1}^{\ell+1} \rho_k \cdot \hat{\boldsymbol{m}}_k, \boldsymbol{z}\rangle$ with randomness r equals $[(\mathsf{cm}_{t_{\boldsymbol{C}}})^{\rho_1} \cdot (\mathsf{cm}_{t_{\boldsymbol{AB}}})^{\rho_{\ell+1}}]^c \cdot \mathsf{cm}_P$.
10: $\mathcal{V}$ accepts if and only if the checks in steps 8 and 9 succeed.

polynomial coefficients $\{c_{2,j}, c_{1,j}, c_{0,j} \mid j \in [\ell]\}$ and $t_{\boldsymbol{AB}}$. Define

$$\begin{aligned} \boldsymbol{\pi} = \big(&c_{2,1},\, c_{1,1},\, c_{0,1},\, c_{2,2},\, \ldots, \\ &c_{2,\ell},\, c_{1,\ell},\, c_{0,\ell},\, t_{\boldsymbol{AB}}\big)^\top \in \mathbb{F}^{3\ell+1}. \end{aligned} \tag{C.2}$$

Then, define the matrix $\boldsymbol{M} \in \mathbb{F}^{(\ell+1)\times(3\ell+1)}$ as

$$\boldsymbol{M}\boldsymbol{\pi} = \begin{pmatrix} t_{\boldsymbol{C}} \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \text{where } \boldsymbol{M} = \begin{pmatrix} \boldsymbol{m}_1 \\ \vdots \\ \boldsymbol{m}_\ell \\ \boldsymbol{m}_{\ell+1} \end{pmatrix}, \text{ with} \tag{C.3}$$

- $\langle \boldsymbol{m}_1, \boldsymbol{\pi}\rangle = t_{\boldsymbol{C}}$ encoding the check of round $j = 1$;
- $\langle \boldsymbol{m}_j, \boldsymbol{\pi}\rangle = 0$ encoding the check of round $j = 2, \ldots, \ell$;
- $\langle \boldsymbol{m}_{\ell+1}, \boldsymbol{\pi}\rangle = 0$ encoding the final check.

Having introduced the required machinery, we now state the formal relation for aggregated sumchecks.

Definition C.2 (Aggregated Sumcheck Relation)**.** *The aggregated sumcheck relation $\mathcal{R}_{\text{agg-sum}}$ is defined over public inputs x and witnesses w. We have $(x; w) \in \mathcal{R}_{\text{agg-sum}}$ if and only if*

$$x = \left(\{\mathsf{cm}_{\boldsymbol{c_j}}\}_{j\in[\ell]}, \mathsf{cm}_{t_{\boldsymbol{C}}}, \mathsf{cm}_{t_{\boldsymbol{AB}}}, \boldsymbol{r}_3\right),$$

$$w = \left(\{\boldsymbol{c_j}\}_{j\in[\ell]}, \{r_{\boldsymbol{c}_j}\}_{j\in[\ell]}, t_{\boldsymbol{C}}, r_{\boldsymbol{C}}, t_{\boldsymbol{AB}}, r_{\boldsymbol{AB}}\right)$$

such that $\boldsymbol{M}\boldsymbol{\pi} = (t_{\boldsymbol{C}}, 0, \cdots, 0)^\top$, and for all $j \in [\ell]$: $\mathsf{cm}_{\boldsymbol{c}_j} = \mathsf{Com}(\boldsymbol{c}_j, r_{\boldsymbol{c}_j})$, $\mathsf{cm}_{t_{\boldsymbol{C}}} = \mathsf{Com}(t_{\boldsymbol{C}}, r_{\boldsymbol{C}})$, $\mathsf{cm}_{t_{\boldsymbol{AB}}} = \mathsf{Com}(t_{\boldsymbol{AB}}, r_{\boldsymbol{AB}})$, where $\boldsymbol{\pi}$ and $\boldsymbol{M}$ are defined as in (C.2) *and* (C.3)*, respectively.*

In order to formulate a zero-knowledge proof for $\mathcal{R}_{\text{agg-sum}}$ we note that $\mathcal{V}$ can multiply each row of $\boldsymbol{M}$ by a random scalar $\rho_k \in \mathbb{F}$ and sum the results [20, Lemma 3]. This reduces to $\mathcal{P}$ wanting to prove to $\mathcal{V}$ that

$$\left\langle \sum_{k=1}^{\ell+1} \rho_k \cdot \boldsymbol{m}_k, \boldsymbol{\pi} \right\rangle = \rho_1 t_{\boldsymbol{C}}. \tag{C.4}$$

Soundness then follows from the Schwartz–Zippel lemma.

Normally, (C.4) would reduce to a single $\text{proof}_{\log}$-of-dot-product. However, $\boldsymbol{\pi}$ cannot be sent all at once, since after each polynomial that $\mathcal{P}$ shares, $\mathcal{V}$ responds with a random challenge r_j. To handle this, $\mathcal{P}$ combines the three coefficients of each polynomial into a single vector commitment $\mathsf{cm}_{\boldsymbol{c}_j}$ (cf. step 3 of Algorithm 2), and we can then follow a slightly adapted version of Hyrax's proof-of-sumcheck protocol, which is designed to address exactly this issue.

Let $\hat{\boldsymbol{m}}_k$ denote $\boldsymbol{m}_k$ without the last element, i.e., a vector in $\mathbb{F}^{3\ell}$. Similarly, let $\hat{\boldsymbol{\pi}} \in \mathbb{F}^{3\ell}$ be $\boldsymbol{\pi}$ without the last entry. Note that the left-hand-side of (C.4) can then be rewritten as $\left\langle \sum_{k=1}^{\ell+1} \rho_k \cdot \hat{\boldsymbol{m}}_k, \hat{\boldsymbol{\pi}} \right\rangle - \rho_{\ell+1} t_{\boldsymbol{AB}}$. Proving (C.4) is therefore equivalent to proving the following statement:

$$\left\langle \sum_{k=1}^{\ell+1} \rho_k \cdot \hat{\boldsymbol{m}}_k, \hat{\boldsymbol{\pi}} \right\rangle = \rho_1 t_{\boldsymbol{C}} + \rho_{\ell+1} t_{\boldsymbol{AB}}. \tag{C.5}$$

To reiterate, we must check a number of sumcheck relations, expressed in matrix-vector form in (C.3). By Schwartz-Zippel, it suffices to check relation (C.4). Rewriting this yields expression (C.5), which is proved using Algorithm 7. A security proof of this algorithm can be found in Appendix E.

C.4. Training Procedure

Algorithm 8 describes the mini-batch gradient descent training procedure. Compared to the standard approach we explicitly construct the ReLU activation masks $\boldsymbol{T}_i^{\prime(\ell)}$ and include intermediate scaling steps.

Appendix D. Exact sumcheck relations

To evaluate the matrix operations in Algorithm 5, we rely on the sum-check protocol. Specifically, for each relevant equation we run adaptations of Algorithm 2, following the constructions of Section 4.3. The correspondence between sum-check formulations and matrix equations in Algorithm 5 follows by construction. To ensure multilinearity and to select matrices corresponding to the correct layers and iterations, we introduce suitable indicator functions β and γ_*. Recall that $\widetilde{\beta}(\boldsymbol{x}, \boldsymbol{y})$ denotes the MLE of the indicator function $\beta(\boldsymbol{x}, \boldsymbol{y})$, which evaluates to 1 if and only if $\boldsymbol{x} = \boldsymbol{y}$.

Algorithm 8 Complete Local Training Round with ReLU and MSE.

Input:

- Dataset $D = (\boldsymbol{X}, \boldsymbol{Y})$, shuffled with permutation ψ and then split in N_{batch} batches $\{B_i\}_{i\in[N_{\text{batch}}]} = \{(\boldsymbol{X}_i, \boldsymbol{Y}_i)\}$ of size S_{batch};
- Scaling factor s; learning rate η; number of epochs E;
- Initial weights $\{\boldsymbol{W}_0^{(l)}\}_{\ell\in[L]}$.

for $i = 1, \ldots, N_{\text{batch}} \cdot E$ **do**
  $\boldsymbol{U}_i^{(0)} = \boldsymbol{X}_{(i-1) \pmod{N_{\text{batch}}}+1}$
  for $\ell = 1, \ldots, L$ **do** ▷ Begin forward pass
    $\boldsymbol{T}_i^{(\ell)} = \boldsymbol{W}_{i-1}^{(\ell)} \cdot \boldsymbol{U}_i^{(\ell-1)}$
    $\boldsymbol{T}_i'^{(\ell)} = \mathbf{1}_{x>0}(\boldsymbol{T}_i^{(\ell)})$
    $\hat{\boldsymbol{T}}_i^{(\ell)} = \boldsymbol{T}_i^{(\ell)} \gg s$
    $\boldsymbol{U}_i^{(\ell)} = \boldsymbol{T}_i'^{(\ell)} \circ \hat{\boldsymbol{T}}_i^{(\ell)}$ ▷ ReLU activation
  end for
  $\boldsymbol{R}_i^{(L+1)} = \left\lfloor \frac{2\cdot 2^s}{S_{\text{batch}}} \right\rfloor (\boldsymbol{U}_i^{(L)} - \boldsymbol{Y}_i)$ ▷ MSE loss function
  $\hat{\boldsymbol{R}}_i^{(L+1)} = \boldsymbol{R}_i^{(L+1)} \gg s$
  for $\ell = L, \ldots, 1$ **do** ▷ Begin backward pass
    $\overline{\boldsymbol{G}}_i^{(\ell)} = (\hat{\boldsymbol{R}}_i^{(\ell+1)} \circ \boldsymbol{T}_i'^{(\ell)}) \cdot (\boldsymbol{U}_i^{(\ell-1)})^\top$
    $\boldsymbol{R}_i^{(\ell)} = (\boldsymbol{W}_{i-1}^{(\ell)})^\top \cdot (\hat{\boldsymbol{R}}_i^{(\ell+1)} \circ \boldsymbol{T}_i'^{(\ell)})$
    $\hat{\boldsymbol{R}}_i^{(\ell)} = \boldsymbol{R}_i^{(\ell)} \gg s$
  end for
  for $\ell = 1, \ldots, L$ **do** ▷ Begin weight update
    $\hat{\boldsymbol{G}}_i^{(\ell)} = \lfloor 2^s \cdot \eta \rfloor \cdot \overline{\boldsymbol{G}}_i^{(\ell)} \gg 2s$
    $\boldsymbol{W}_i^{(\ell)} = \boldsymbol{W}_{i-1}^{(\ell)} - \hat{\boldsymbol{G}}_i^{(\ell)}$
  end for
end for

The sum-check formulation for step 2 is given by

$$\widetilde{\boldsymbol{T}}(\boldsymbol{r}_4, \boldsymbol{r}_1, \boldsymbol{r}_2) = \sum_{\boldsymbol{r}_2', \boldsymbol{r}_3, \boldsymbol{r}_4', \boldsymbol{r}_4'' \in \{0,1\}^*} \widetilde{\beta}(\boldsymbol{r}_2, \boldsymbol{r}_2')\, \widetilde{\gamma}_1(\boldsymbol{r}_4, \boldsymbol{r}_4', \boldsymbol{r}_4'') \cdot \widetilde{\boldsymbol{W}}(\boldsymbol{r}_4'', \boldsymbol{r}_1, \boldsymbol{r}_3)\, \widetilde{\boldsymbol{T}'}(\boldsymbol{r}_4', \boldsymbol{r}_3, \boldsymbol{r}_2')\, \widetilde{\hat{\boldsymbol{T}}}(\boldsymbol{r}_4', \boldsymbol{r}_3, \boldsymbol{r}_2'). \quad \text{(D.1)}$$

Here, $\gamma_1(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ is the indicator function that evaluates to 1 if and only if $\boldsymbol{x} = \boldsymbol{y} + 1 = \boldsymbol{z}$ and $x \neq 0 \pmod{L+2}$. We write $\boldsymbol{y}+1$ for the binary representation of $1+\sum_i 2^{i-1} y_i$.

Analogously, the sum-check formulation for step 4 is

$$\widetilde{\overline{\boldsymbol{G}}}(\boldsymbol{r}_4, \boldsymbol{r}_1, \boldsymbol{r}_2) = \sum_{\boldsymbol{r}_1', \boldsymbol{r}_2', \boldsymbol{r}_3, \boldsymbol{r}_4', \boldsymbol{r}_4'', \boldsymbol{r}_4''' \in \{0,1\}^*} \widetilde{\beta}(\boldsymbol{r}_1, \boldsymbol{r}_1') \cdot \widetilde{\beta}(\boldsymbol{r}_2, \boldsymbol{r}_2')\, \widetilde{\gamma}_2(\boldsymbol{r}_4, \boldsymbol{r}_4', \boldsymbol{r}_4'', \boldsymbol{r}_4''')\, \widetilde{\hat{\boldsymbol{R}}}(\boldsymbol{r}_4''', \boldsymbol{r}_1', \boldsymbol{r}_3) \cdot \widetilde{\boldsymbol{T}'}(\boldsymbol{r}_4'', \boldsymbol{r}_1', \boldsymbol{r}_3)\, \widetilde{\boldsymbol{T}'}(\boldsymbol{r}_4'', \boldsymbol{r}_2', \boldsymbol{r}_3)\, \widetilde{\hat{\boldsymbol{T}}}(\boldsymbol{r}_4', \boldsymbol{r}_2', \boldsymbol{r}_3). \quad \text{(D.2)}$$

Here, $\gamma_2(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{w})$ evaluates to 1 if and only if $\boldsymbol{x} = \boldsymbol{y} + 1 = \boldsymbol{z} = \boldsymbol{w} - 1$ and $0 < x < L + 1 \pmod{L+2}$.

Finally, the sum-check formulation for Step 5 is

$$\widetilde{\hat{\boldsymbol{R}}}(\boldsymbol{r}_4, \boldsymbol{r}_1, \boldsymbol{r}_2) = \sum_{\boldsymbol{r}_2', \boldsymbol{r}_3, \boldsymbol{r}_4', \boldsymbol{r}_4'' \in \{0,1\}^*} \widetilde{\beta}(\boldsymbol{r}_2, \boldsymbol{r}_2')\, \widetilde{\gamma}_3(\boldsymbol{r}_4, \boldsymbol{r}_4', \boldsymbol{r}_4'') \cdot \widetilde{\boldsymbol{W}}(\boldsymbol{r}_4', \boldsymbol{r}_3, \boldsymbol{r}_1)\, \widetilde{\hat{\boldsymbol{R}}}(\boldsymbol{r}_4'', \boldsymbol{r}_3, \boldsymbol{r}_2')\, \widetilde{\boldsymbol{T}'}(\boldsymbol{r}_4', \boldsymbol{r}_3, \boldsymbol{r}_2'). \quad \text{(D.3)}$$

Here, $\gamma_3(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ evaluates to 1 if and only if $\boldsymbol{x} = \boldsymbol{y} = \boldsymbol{z} - 1$ and $1 < x < L + 1 \pmod{L+2}$.

Appendix E. Security Lemmas and Proofs

In this section we present the lemmas for the constructions used in FALAFEL, including precise soundness formulations. Throughout, we use the notation $a^{\times n} = (a, a, \ldots, a) \in \mathbb{N}^n$ introduced in Section 2.

Lemma E.1. *Algorithm 1 is complete, $\boldsymbol{k}$-out-of-$\boldsymbol{N}$ special sound, and honest-verifier perfect zero-knowledge, where*

$$\boldsymbol{k} = (3^{\times \log(S_{\text{batch}} \cdot \max(d, n_L))},\ 2,\ 3^{\times \log(N_{\text{samples}} \cdot d)},\ 2,\ 3^{\times \log(N_{\text{samples}} \cdot n_L)},\ 2),$$

and $\boldsymbol{N} = (p-1, \ldots, p-1)$.

Proof. **Completeness.** Assume $(x, w) \in \mathcal{R}_{\mathsf{shf\text{-}spl}}$. Then, for any $z \in \mathbb{F}^*$ chosen by $\mathcal{V}$, the values t_1, t_2 computed by $\mathcal{P}$ satisfy (by definition of $\mathcal{R}_{\mathsf{shf\text{-}spl}}$ and $\boldsymbol{z}'_X, \boldsymbol{z}'_Y$) the equalities required in the three proof$_{\log}$-of-dot-product subprotocols. Completeness thus follows from the completeness of the proof$_{\log}$-of-dot-product protocol.

Special Soundness. The special soundness parameters follow from sequential composition of the sub-protocols [39, Lemma 3.1]. The protocol invokes three instances of the proof$_{\log}$-of-dot-product protocol of [20, A.3]. The first one is $(3^{\times \log(S_{\text{batch}} \cdot \max(d, n_L))}, 2)$ special sound, the second one $(3^{\times \log(N_{\text{samples}} \cdot d)}, 2)$ special sound and the last one $(3^{\times \log(N_{\text{samples}} \cdot n_L)}, 2)$ special sound. The initial challenge z does not contribute to the special soundness parameters.

Honest-Verifier Perfect Zero-Knowledge. The simulator samples the verifier challenge z and uniformly samples the commitments from steps 2 and 3. Although in a real execution these commitments are deterministically related via the witness and z, this dependency is hidden by the perfect hiding property of the commitment scheme and is never checked outside the three proof-of-dot-product sub-protocols. The simulator therefore invokes the simulators for these sub-protocols. Since consistency is enforced only within them and they are HVPZK, their composition yields a perfectly simulated transcript. □

Lemma E.2. *Algorithm 2 is complete, $\boldsymbol{k}$-out-of-$\boldsymbol{N}$ special sound and honest-verifier perfect zero-knowledge, where*

$$\boldsymbol{k} = (\kappa_{\mathsf{poly\text{-}eval}}^{\times 3},\ \kappa_{\mathsf{prod}},\ \kappa_{\mathsf{agg\text{-}sum}}),$$
$$\boldsymbol{N} = (\lambda_{\mathsf{poly\text{-}eval}}^{\times 3},\ \lambda_{\mathsf{prod}},\ \lambda_{\mathsf{agg\text{-}sum}}),$$

and where

$$\kappa_{\mathsf{agg\text{-}sum}} = (2, 2), \qquad \lambda_{\mathsf{agg\text{-}sum}} = (p(\ell+1), p),$$
$$\kappa_{\mathsf{poly\text{-}eval}} = (3^{\times \ell}, 2), \qquad \lambda_{\mathsf{poly\text{-}eval}} = (p-1)^{\times(\ell+1)},$$
$$\kappa_{\mathsf{prod}} = 2, \qquad \lambda_{\mathsf{prod}} = p.$$

Proof. **Completeness.** Follows from the properties of the sum-check protocol and the completeness of the sub-protocols used.

Special Soundness. The special soundness parameters follow from sequential composition of the sub-protocols [39, Lemma

3.1]. The only non-obvious parts concern the challenges $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ selected at the start of the protocol, and the component-wise choice of the challenge $\boldsymbol{r}_3$ in the for-loop (step 3). These challenges merely fix $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ and $\boldsymbol{r}_3$ as public input to later sub-protocols and are therefore not reflected in the special soundness parameters.

We refer to Theorem E.4 for a discussion of the sub-optimality of these parameters.

Honest-Verifier Perfect Zero-Knowledge.

The simulator proceeds by randomly sampling all challenges and commitments in steps 1, 2, 3 and 5. It then invokes the simulators for each of the sub-protocols. The concatenation yields a perfectly simulated transcript by the HVPZK of each sub-protocol. □

Lemma E.3. *Algorithm 3 is complete,* $(n^2+2, 3^{\times \log(n)}, 2, qn^2, 3, 2, 4^{\times \log(qn^2)})$ *special sound and honest-verifier perfect zero-knowledge.*

Proof. **Completeness.** We recall that our proof of scaling is an adaptation of the Aggregated Logarithmic Range Proofs from Bulletproofs. Since that protocol is complete, we only argue completeness for the introduced modifications

For steps 1–4, completeness follows by inspection. In step 1, we first shift the witnesses in A and B to ensure non-negativity, while keeping the relation between $\boldsymbol{A}$ and $\boldsymbol{B}$ intact. Then, the construction of $\mathsf{cm}_{\boldsymbol{a}}$, and cm_v is analogous to Bulletproofs. Note, that we only require the bits of $\boldsymbol{B}$, since the bits for $\boldsymbol{A}$ are a subset of those.

In particular, when the prover computes valid bit decompositions and corresponding commitments in steps 1 and 3, the commitment cm_v opens to the intended linear combination of the entries of $\boldsymbol{A}$ and $\boldsymbol{B}$. Hence, the $\text{proof}_{\log}$-of-dot-product in step 4 verifies by completeness of the underlying protocol.

The remaining deviation is the extension of the polynomial $r(X)$ in step 6. For an honest prover, this additional term enforces that the bits encoding a_{ij} coincide with the shifted most significant bits of b_{ij}. All subsequent checks are adapted accordingly, and completeness follow from inspection and the completeness of the Aggregated Logarithmic Range proof on which construction is based.

Special Soundness. We consider the soundness properties of the $\text{proof}_{\log}$-of-dot-product and the range proof separately, then combine them into the desired expression for Algorithm 3 as a whole. The $\text{proof}_{\log}$-of-dot-product is $(3^{\times \log(n)}, 2)$ special sound as discussed in Appendix C.1.

The range proof part adapts the Bulletproofs aggregated logarithmic range proof with $m = n^2$ commitments to q-bit values (instead of m commitments to n-bit values). In particular, our modification only introduces extra terms without changing polynomial degrees nor the extraction requirements they provide. While the original Bulletproofs paper [13, Thm 2, Appendix D] constructs an extractor for witness-extended emulation, we find that the arguments translate directly to our machinery of special soundness by simply substituting our parameters into their expressions. That is, the extractor samples qn^2 values of y, n^2+2 values of z and 3 values of x. As a subroutine, the constructed extractor runs the IPA extractor on each of these transcripts. Theorem 1 of [13] claims that their inner-product argument satisfies witness-extended emulation. Again, a closer inspection reveals that their proof actually establishes that the core $(2\log(n)+1)$-round subprotocol is $(4, \ldots, 4)$-special-sound. The full inner-product argument adds an extra round and can thus be shown to be $(2, 4, \ldots, 4)$-special-sound. Note that our adaptation changes the order of the challenges y and z. Putting this all together, we find that the range proof part of Algorithm 3 is $(n^2+2, qn^2, 3, 2, 4, \ldots, 4)$-out-of-$(p-1, \ldots, p-1)$ special sound.

Although the $\text{proof}_{\log}$-of-dot-product is invoked as a sub-protocol within Algorithm 3, we note that it is essentially independent. The only constraint is that it has to occur after fixing the challenge z, but it does not require z to be varied for the extractor to work. Therefore, we can simply inject the special soundness parameters of the $\text{proof}_{\log}$-of-dot-prod directly after the first entry of the range proof's special soundness parameters, yielding special soundness parameters we set out to prove.

We refer to Theorem E.4 for a discussion of the sub-optimality of these parameters.

Honest-Verifier Perfect Zero-Knowledge. We construct a simulator $\mathcal{S}$ by making minor adaptations to the aggregated range proof simulator from Bulletproofs [13, Theorem 2, Appendix D]. First, $\mathcal{S}$ uniformly samples the additional commitment cm_v. Second, $\mathcal{S}$ invokes the $\text{proof}_{\log}$-of-dot-product simulator on $(\prod_{j=1}^{n} \mathsf{cm}_{\boldsymbol{b}_j}^{z^{(j-1)n}}) \cdot (z^{n^2} \prod_{j=1}^{n} \mathsf{cm}_{\boldsymbol{a}_j}^{z^{(j-1)n}}, \boldsymbol{z}^n, \mathsf{cm}_v)$. Lastly, the computation of $\mathsf{cm}_{\boldsymbol{s}}$ and cm_{t_1} is handled by using our modified verification equations. It follows that the output of $\mathcal{S}$ is indistinguishable from a real conversation. □

Remark E.4. *We observe that the special-soundness analysis in Lemma E.2 and Lemma E.3 is somewhat suboptimal. Specifically, it implies a knowledge extractor whose running time and knowledge error parameters are not optimal. This suboptimality arises because our protocols execute several subprotocols sequentially, even though these subprotocols are independent and could be executed in parallel. Exploiting this parallelism would immediately reduce the number of rounds in the composed protocols and enable more efficient knowledge extraction, operating on a tree of smaller depth. Such a parallel approach can be rigorously analyzed using the generalized special-soundness framework of [40], which would improve both the knowledge error and the extractor's running time. However, since the practical gains are limited in our setting, we decided for the sequential construction and simpler analysis presented above.*

Lemma E.5. *Algorithm 6 is complete,* $(3^{\times \ell}, 2)$*-out-of-*$(p-1, \ldots, p-1)$ *special sound and honest-verifier perfect zero-knowledge.*

Proof. **Completeness.** We sketch a proof and refer to [20, Lemma 5] for details. By construction, we have $\langle \boldsymbol{LY}, \boldsymbol{R} \rangle = \widetilde{\boldsymbol{Y}}(\boldsymbol{r}_1, \boldsymbol{r}_2)$. Moreover, $\mathsf{cm}_{\boldsymbol{t}'}$ is a commitment to $\boldsymbol{LY}$. These

equalities together with the $\text{proof}_{\log}$-of-dot-product in step 3 show that $\widetilde{\boldsymbol{Y}}(\boldsymbol{r}_1, \boldsymbol{r}_2) = \langle \boldsymbol{LY}, \boldsymbol{R} \rangle = \langle \boldsymbol{t}', \boldsymbol{R} \rangle = t$. Completeness follows.

Special Soundness. Let $\mathcal{E}'$ be an extractor for the $\text{proof}_{\log}$-of-dot-product invoked in step 3. By Appendix C.1, this protocol is $(3, \ldots, 3, 2)$-out-of-$(p-1, \ldots, p-1)$ special sound. We extend $\mathcal{E}'$ to an extractor $\mathcal{E}$ for Algorithm 6 with the same special soundness parameters.

To see this, note that a witness for the $\text{proof}_{\log}$-of-dot-product consists of a vector $\boldsymbol{t'}$ with associated randomness $\rho_{t'}$ and a scalar t with associated randomness ρ_t. Note that the vector $\boldsymbol{L}$ is public. By solving linear systems of equations, $\mathcal{E}$ can construct a matrix $\boldsymbol{Y}$ and a vector $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_{2^\ell - 1})$ satisfying $\boldsymbol{LY} = \boldsymbol{t'}$ and $\langle \boldsymbol{L}, \boldsymbol{\rho} \rangle = \rho_{t'}$. It follows immediately that $\left(\boldsymbol{Y}, t, \{\rho_k\}_{k=0}^{2^\ell - 1}, \rho_t\right)$ is a witness for $\mathcal{R}_{\mathsf{poly\text{-}eval}}$.

Honest-Verifier Perfect Zero-Knowledge. Follows from the construction, since the underlying $\text{proof}_{\log}$-of-dot-product is honest-verifier perfect zero-knowledge. See [20, Lemma 5] for more details. □

Lemma E.6. *Algorithm 7 is complete,* $(2,2)$*-out-of-*$(p(\ell+1), p)$ *special sound and honest-verifier perfect zero-knowledge.*

Proof. **Completeness.** Algorithm 7 is based on proof-of-sumcheck in Hyrax, with just minor adaptations. Specifically, Algorithm 7 differs only in the fact that we require fewer polynomial coefficients. Therefore, the proof for completeness carries over directly from the proof presented in Hyrax [20, A.4]

Special Soundness. We show that there exists an algorithm which, given a tree of four accepting transcripts, outputs a valid witness for the relation $\mathcal{R}_{\mathsf{agg\text{-}sum}}$. Concretely, suppose we are given the following $(2,2)$-tree of accepting transcripts:

$$\begin{aligned}
&\left(\{\mathsf{cm}_{\boldsymbol{d}_j}\}, (\rho_1, \ldots, \rho_{\ell+1}), \mathsf{cm}_P, c^{(1)}, \left(\boldsymbol{z}^{(1)}, \{r^{(1)}_{\boldsymbol{z}_j}\}, r^{(1)}\right)\right),\\
&\left(\{\mathsf{cm}_{\boldsymbol{d}_j}\}, (\rho_1, \ldots, \rho_{\ell+1}), \mathsf{cm}_P, c^{(2)}, \left(\boldsymbol{z}^{(2)}, \{r^{(2)}_{\boldsymbol{z}_j}\}, r^{(2)}\right)\right),\\
&\left(\{\mathsf{cm}_{\boldsymbol{d}_j}\}, (\bar{\rho}_1, \ldots, \bar{\rho}_{\ell+1}), \mathsf{cm}_{\bar{P}}, c^{(3)}, \left(\boldsymbol{z}^{(3)}, \{r^{(3)}_{\boldsymbol{z}_j}\}, r^{(3)}\right)\right),\\
&\left(\{\mathsf{cm}_{\boldsymbol{d}_j}\}, (\bar{\rho}_1, \ldots, \bar{\rho}_{\ell+1}), \mathsf{cm}_{\bar{P}}, c^{(4)}, \left(\boldsymbol{z}^{(4)}, \{r^{(4)}_{\boldsymbol{z}_j}\}, r^{(4)}\right)\right),
\end{aligned}$$

where $c^{(1)} \neq c^{(2)}$ and $c^{(3)} \neq c^{(4)}$.

From the first two transcripts, define $\boldsymbol{z} := (c^{(1)} - c^{(2)})^{-1}(\boldsymbol{z}^{(1)} - \boldsymbol{z}^{(2)})$ and $r_{\boldsymbol{z}_j} := (c^{(1)} - c^{(2)})^{-1}(r^{(1)}_{\boldsymbol{z}_j} - r^{(2)}_{\boldsymbol{z}_j})$, where $\boldsymbol{z}_j = (z_{2,j}, z_{1,j}, z_{0,j})$ for $\boldsymbol{z} = (z_{2,1}, z_{1,1}, z_{0,1}, \ldots, z_{2,\ell}, z_{1,\ell}, z_{0,\ell})$. By the verification equations in step 8, we have

$$\mathsf{Com}(\boldsymbol{z}_j, r_{\boldsymbol{z}_j}) = \mathsf{cm}_{\boldsymbol{c}_j} \quad \text{for all } j \in [\ell],$$

so $\boldsymbol{z}$ is a vector opening all commitments $\{\mathsf{cm}_{\boldsymbol{c}_j}\}$. Analogous to the above, we can use the latter two transcripts to find a second opening $\bar{\boldsymbol{z}}$ and associated randomness. Since $\boldsymbol{z}$ and $\bar{\boldsymbol{z}}$ both commit to the same witness, the binding property of the commitment scheme yields $\boldsymbol{z} = \bar{\boldsymbol{z}}$.

Since the first transcript is accepting, step 9 implies that $\boldsymbol{z}$ satisfies

$$\left\langle \sum_{k=1}^{\ell+1} \rho_k \hat{\boldsymbol{m}}_k, \boldsymbol{z} \right\rangle = \rho_1 t_{\boldsymbol{C}} + \rho_{\ell+1} t_{\boldsymbol{AB}}. \tag{E.1}$$

Analogously, the third transcript shows that $\boldsymbol{z}$ also satisfies:

$$\left\langle \sum_{k=1}^{\ell+1} \bar{\rho}_k \hat{\boldsymbol{m}}_k, \boldsymbol{z} \right\rangle = \bar{\rho}_1 t_{\boldsymbol{C}} + \bar{\rho}_{\ell+1} t_{\boldsymbol{AB}}. \tag{E.2}$$

Together, (E.1) and (E.2) form a linear system in the unknowns $t_{\boldsymbol{C}}$ and $t_{\boldsymbol{AB}}$. As the $\hat{\boldsymbol{m}}_k$ are public and the values ρ_j and $\bar{\rho}_j$ for $j \in \{1, \ldots, \ell+1\}$ appear in the transcripts, solving this system uniquely determines $t_{\boldsymbol{C}}$ and $t_{\boldsymbol{AB}}$. The corresponding randomness values $r_{\boldsymbol{C}}$ and $r_{\boldsymbol{AB}}$ are obtained analogously from $r^{(1)}, r^{(2)}, r^{(3)}, r^{(4)}$.

Consequently, the algorithm outputs a valid witness

$$\left(\{\boldsymbol{c}_j\}_{j\in[\ell]}, \{r_{\boldsymbol{c}_j}\}_{j\in[\ell]}, t_{\boldsymbol{C}}, r_{\boldsymbol{C}}, t_{\boldsymbol{AB}}, r_{\boldsymbol{AB}}\right),$$

establishing $(2,2)$-special soundness.

Honest-Verifier Perfect Zero-Knowledge. This protocol admits a honest-verifier zero-knowledge simulator that is a straightforward adaptation of the one presented in Hyrax [20, A.4]. For this reason we do not spell it out here. □