Experimenting with Zero-Knowledge Proofs of Training

Sanjam Garg[*] Aarushi Goel[†] Somesh Jha[‡] Saeed Mahloujifar[§]
Mohammad Mahmoody[¶] Guru-Vamsi Policharla[‖] Mingyuan Wang[**]

Abstract

How can a model owner prove they trained their model according to the correct specification? More importantly, how can they do so while preserving the privacy of the underlying dataset and the final model? We study this problem and formulate the notion of *zero-knowledge proof of training (zkPoT)*, which formalizes rigorous security guarantees that should be achieved by a privacy-preserving proof of training. While it is theoretically possible to design zkPoT for any model using generic zero-knowledge proof systems, this approach results in extremely unpractical proof generation times. Towards designing a practical solution, we propose the idea of combining techniques from MPC-in-the-head and zk-SNARKs literature to strike an *appropriate trade-off* between proof size and proof computation time. We instantiate this idea and propose a concretely efficient, novel zkPoT protocol for logistic regression.

Crucially, our protocol is streaming-friendly and does not require RAM proportional to the size of the circuit being trained and, hence, can be adapted to the requirements of available hardware. We expect the techniques developed in this paper to also generally be useful for designing efficient zkPoT protocols for other relatively more sophisticated ML models.

We implemented and benchmarked prover/verifier runtimes and proof sizes for training a logistic regression model using mini-batch gradient descent on a 4 GB dataset of 262,144 records with 1024 features. We divide our protocol into three phases: (1) data-independent offline phase (2) data-dependent phase that is independent of the model (3) online phase that depends both on the data and the model. The total proof size (across all three phases) is less than 10% of the data set size (< 350 MB). In the online phase, the prover and verifier times are under 10 minutes and half a minute respectively, whereas in the data-dependent phase, they are close to one hour and a few seconds respectively.

[*]UC Berkeley sanjamg@berkeley.edu
[†]NTT Research aarushi.goel@ntt-research.com
[‡]University of Wisconsin, Madison jha@cs.wisc.edu
[§]Meta saeedm@meta.com
[¶]University of Virginia mohammad@virginia.edu
[‖]UC Berkeley guruvamsip@berkeley.edu
[**]UC Berkeley mingyuan@berkeley.edu

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 4
- 1.2 Technical Overview . . . 5
- 1.3 Related Works . . . 9
- 1.4 Road Map . . . 9

2 Preliminaries **10**
- 2.1 (Packed) Secret Sharing . . . 10
- 2.2 Machine Learning . . . 11
- 2.3 Secure Multiparty Computation . . . 11
- 2.4 Secure Fix-point Arithmetic Operations . . . 12
- 2.5 (List) Polynomial Commitment . . . 13
- 2.6 Protocols for Sumcheck and Permutation . . . 14

3 Formalizing Proof of Training **16**

4 MPC Protocol for MPC-in-the-Head **18**
- 4.1 Overview . . . 18
- 4.2 Protocol Description . . . 20
  - 4.2.1 Pre-Processing Phase. . . . 20
  - 4.2.2 Online Phase. . . . 21

5 Zero-Knowledge Sub-Routines for Specific Relations **23**
- 5.1 Checking Consistency of Packed Secret Shares . . . 24
- 5.2 Checking Consistency of Correlated Randomness . . . 25
- 5.3 Checking the Range for Truncation Masks . . . 26
- 5.4 Checking Consistency of Data Shares . . . 27

6 Zero-Knowledge Proof of Training for Logistic Regression **29**
- 6.1 Overview . . . 29
- 6.2 Protocol Description . . . 30
- 6.3 Extension: zkPoT for Other Models . . . 34

7 Implementation and Evaluation **35**
- 7.1 Benchmarks . . . 37

1 Introduction

Machine learning models continue to advance at an unprecedented pace, characterized by ever-expanding capabilities and resource demands [TMS+23, ND21]. State-of-the-art models are predominantly developed by organizations possessing extensive resources (e.g., openAI and Google), ranging from computational resources, proprietary data, and intellectual property. In certain instances, model trainers must provide evidence of compliance with particular training specifications, while maintaining the confidentiality of the datasets or the model itself. Furthermore, they might need their evidence of training to be verifiable with fewer resources than were required for training the model.

Consider, for instance, a model proprietor offering the model as a service (e.g., for a loan or credit approval). They may face allegations of neglecting requisite safety measures intended to ensure the fairness of decisions[1] or the privacy of data [nyt]. In such scenarios, the model owner may be compelled to present proof of adhering to the appropriate specifications that satisfy constraints related to safety properties. Another type of evidence that may be required pertains to model ownership, as an entity that trains and provides a model as a service might need to substantiate their claims against accusations of model theft or distillation [TZJ+16] by providing a proof that they trained the model themselves.

In this study, we present the concept of *proof of training (PoT)*, a primitive that enables model trainers to prove the proper execution of training optimization. Specifically, while training a model M on dataset D, the learner concurrently generates a proof π. Subsequently, a verifier equipped with π can verify whether the prover has trained the model according to the specifications of the learning algorithm. A crucial requirement for such a proof system, as exemplified by both aforementioned applications, is *zero-knowledge* [GMR85]. The proof π must preserve the confidentiality of the model M and the dataset D used during training. In applications where PoT aims to demonstrate the privacy of model training (e.g., proving that the model was trained with differentially private stochastic gradient descent (DPSGD), data privacy is paramount, rendering the proof irrelevant if the entire training dataset is exposed. In the case of model ownership, exposing the model or dataset would likewise be self-defeating. If a model holds sufficient value to warrant a proof of ownership, then revealing the model parameters or the training dataset would be undesirable. Consequently, we concentrate on *zero-knowledge* proofs of training that preserve the confidentiality of the model and the associated dataset, which we call *zero-knowledge proof of training* (zkPoT).

At first glance, the notion of zkPoT may appear paradoxical; *how can one substantiate model training without disclosing the model or training set?* The crux lies in the model owner providing a PoT for a model and a dataset to which they have cryptographically committed [Nao90]. Therefore, a proof of training validates a statement of the following nature: "The prover possesses a model M consistent with a commitment c_M, dataset D is consistent with a commitment c_D, and the model M has been trained on dataset D according to training algorithm L." To verify this statement, the verifier receives a proof π, commitments c_M, c_D, and the training algorithm specifications (such as the number of steps, architecture, learning rate, batch size, etc.). The verifier can then proceed to further examine the commitments of the model and the dataset. For example, the verifier may request proof that a query response was generated by a model M consistent with the commitment c_M (e.g., using a proof of inference [LXZ21a, LKKO20, FQZ+21]) or a proof that the dataset D linked to commitment c_D has certain properties (e.g., containing specific examples, using commitments with partial selective opening [BHK12]).

[1]As a concrete example, there is well-documented evidence[exa] showing that black homeowners typically have higher interest rates than white homeowners, even when they have substantially higher incomes.

1.1 Our Contributions

In this work, we formulate the notion of zero-knowledge proof of training (zkPoT) as a cryptographic proof primitive. This proof system enables the learner to prove that the training was done according to the desired specification without leaking the dataset or the final model.

Over the years, significant research has resulted in zero-knowledge proof systems that are capable of generating privacy-preserving proofs for any generic computation, and theoretically, one could consider using such proof systems for generating a proof of ML training. However, these generic proofs typically either require a significant overhead in the proof generation time or produce a gigantic proof. For example, succinct zero-knowledge non-interactive argument (zk-SNARK) [Kil92, Mic94, BCC⁺17] result in extremely short proofs, but incur a massive (of the order of at least a thousand times) overhead in the proof generation time as compared to the training time, rendering them essentially infeasible for any reasonably sophisticated training task. On the other hand, the MPC-in-the-head style techniques [IKOS07] incur minimum overhead in generating the proof, but result in proof sizes (as well as verification time) that are as large as the training circuit, making it infeasible to communicate this proof.

To bridge this gap between theory and practice, in this work, we propose the idea of combining zk-SNARK techniques and the MPC-in-the-head techniques for the purpose of zkPoT. This allows us to achieve an appropriate trade-off between the proof generation time and proof size. In particular, we instantiate our idea and design a concretely efficient zkPoT for logistic regression. In detail, assuming we have a dataset with N records and D features each, our proof size only depends linearly on N and is independent of D. This technique enables the possibility of working with massive datasets.

Another significant advantage of our approach is its streaming-friendliness. Unlike most zkSNARKs, our proof generation does not require holding the entire computation trace in main memory (RAM). Instead, it can be loaded in from secondary memory as and when required. As a result, there is no fundamental memory limit to how large a model and data set can be trained using our approach, making it more compatible with ML training tasks.

Implementation and Evaluation. We test our protocol for logistic regression and are able to generate zero-knowledge proofs of training with reasonable efficiency. For the sake of exposition, we find it most instructive to divide our protocol into three phases — Offline Phase (independent of data and model), Data Checks Phase (only dependent on data), and Online Phase (dependent both on the data and model). These provide insights into the bottlenecks of our protocol and are also a meaningful separation as the offline phase can be carried out before the data is even available and data checks phase can be repurposed for proving properties about the dataset and for training a different model.

Here, we provide prover/verifier runtimes and proofs sizes for training a logistic regression model using mini-batch gradient descent on a data set with 262,144 entries, and 1024 features each (amounting to 4 GB in size). The offline prover/verifier time have not been benchmarked, see Section 7 for details on how proof size was estimated. We implemented a prototype of our zkPoT algorithm in Rust and benchmarked the prover/verifier run time and proof sizes.

Phase	Prover (s)	Verifier (s)	Size (MB)
Online	518	24	196
Data Checks	3690	2.5	5.3
Offline	-	-	< 140
Total	-	-	< 350

To put our results in context, the time taken to carry out the actual training is roughly 1 s. However, when training is done over a larger 128-bit finite field that our protocol requires, instead of rust's native `f64` data type, it takes roughly 11.5 s. We note that comparing the overhead of generating a proof of training against both the above times is *meaningful* as the former gives the overhead on top of optimized training algorithms, whereas the latter provides insight into how much overhead is introduced by our techniques beyond the seemingly inherent overhead introduced by moving to larger finite fields that are not natively supported by modern CPU architectures. This large cryptographic overhead[2] is typical of zero-knowledge proofs even for protocols that were specially designed for machine learning circuits. For instance, [LXZ21b] designed a highly optimized proof of inference for CNNs and had similar cryptographic overheads.

Moreover, using an off-the-shelf zkSNARK such as Groth16 [Gro16] is completely infeasible as it would require massive amounts of memory. In fact, prior work [BFH+20] that attempted to do this for a linear model ran out of memory for a training data set of just 2000 points. Our speedup is mainly due to the use of fast hash-based proofs and *completely* avoiding elliptic curve operations. Although our proof sizes are not as small in comparison to zkSNARKs, we argue that this is a favorable trade-off in many situations where the verifier is reasonably powerful, albeit still sub-linear.

Real World Applicability. Although logistic regression is a relatively simple model, a zkPoT for logistic regression is already quite useful in many scenarios. For instance, logistic regression plays an important role in the context of foundation models trained on vast datasets. The success of these models allows for the fine-tuning of smaller networks on top of the feature extractors, even when working with smaller datasets [RKH+21]. In such cases, it is well-motivated for model owners to generate a zkPoT for proving that *the fine-tuning was done honestly* w.r.t. a commitment to the output of the foundation model.

Moreover, logistic regression is only the first step. Developing practical zkPoTs for more sophisticated models is an exciting future research direction, and we expect the techniques that we develop in this work to be generally applicable to more sophisticated models. We refer the readers to more discussion on this in Section 1.2.

1.2 Technical Overview

As discussed in the previous section, our objective is to give a *sound, zero-knowledge* proof to convince the verifier that the final model is the output of an *honest training process* under some committed data set (refer to Section 3 for formal definitions).

Challenges with Existing Approaches. We start by discussing why naïvely using some common existing approaches fail to give a practically feasible solution for our task.

(I) zkSNARKs: One of the most popular types of zero-knowledge proof systems are zero-knowledge succinct non-interactive arguments of knowledge (or zkSNARKs) [Mic94, BCC+17]. zkSNARKs allow a prover to generate a *small proof* of honest computation. Over the last few years, zkSNARKs have been extensively studied and have led to concretely-efficient proof systems for several real-world applications [BCG+14, ZkR21, KGC+18, ZFZS20, RPX+22, GHH+23]. In fact, zkSNARKs have already been successfully implemented in some cases for proof of inference [LKKO20, LXZ21a, FQZ+21].

In general, while zkSNARKs enable extremely fast verification of zero-knowledge proofs, the proof generation time is significantly slower. This is primarily due to the use of cryptographic operations, which are several orders of magnitude more expensive than the computation being proved, resulting in very large

[2]That is the ratio between the time to generate the proof and the time to merely do the training.

cryptographic overhead. In particular, due to the massive amount of data used in training, and the size of the models that are being trained, the amount of computation involved in training ML models is very large. Introducing large cryptographic overheads on top of this would make it completely infeasible.

Additionally, most of the existing zkSNARKs would require the prover to load the *entire trace of computation* in the memory. That is, the prover needs to first do the training in the entirety, *keep track of all the intermediate values computed during training*, then load them all into the memory and perform some computation on it (e.g., fast Fourier transform) to produce a proof. This highly undesirable feature is a deal-breaker for applying existing zkSNARKs.[3]

Finally, another difficulty in applying zkSNARKs to ML training is type mismatch. All zkSNARKs work by representing the given computation as an arithmetic circuit[4] or a set of constraints over some finite field. In contrast, ML training is typically done over real-valued data. Therefore, to use zkSNARKs, one must translate an algorithm for real numbers into an arithmetic circuit. However, a typical training task involves many *non-arithmetic operations*. For instance, if one uses a field element to represent a real number with a fixed precision,[5] a truncation operation must be applied after each multiplication to restore the precision. Therefore, translating these non-native operations into arithmetic operations introduces a significant overhead. As a result, translating such computations into an arithmetic circuit and then computing a zkSNARK proof over it (using existing approaches), is prohibitively expensive.

(II) MPC-in-the-head: The other most common proof generation technique that yields efficient proof systems is one based on MPC-in-the-head [IKOS07]. MPC-in-the-head is a generic approach for building zero-knowledge proofs from secure multiparty computation protocols. Recent works [AHIV17, GMO16, CDG+17, KKW18] have shown that this technique can be used to design proof systems with efficient proof generation.

The advantage of this approach in our context is that efficient MPC protocols for non-arithmetic operations exist. For instance, if parties hold secret shares $[\![a]\!]$ of a and wish to truncate it by m bits, they could reconstruct $a + r$ in the clear using some random mask r and locally compute $(a + r) \mod 2^m$ to correctly cancel the last m bits of a(refer to Section 2.4 for details). Crucially, these non-arithmetic operations can be done in a distributed manner in MPC protocols without translating into arithmetic operations. Consequently, these non-arithmetic operations no longer pose a significant barrier to the prover's run time.

A disadvantage of the MPC-in-the-head approach, however, is that the proof size is large. Indeed, the ZK proof solely generated using the classical MPC-in-the-head approach has a proof size linear in the size of the computation. This is in sharp contrast to zkSNARKs, where proof size is sublinear in the size of the computation. This is far from ideal for massive computations, such as ML training. Indeed, communicating proofs whose size is proportional to the computation involved in ML training would be nearly impossible over most networks.

A New Framework. To overcome the barriers presented by existing approaches, we propose a *new framework* for the efficient generation of succinct zero-knowledge proofs. Conceptually, our framework can be described as follows. Suppose we want to generate a proof for some computation (represented by a circuit C), consisting of two types of gates/operations (e.g., arithmetic and non-arithmetic operations). Our idea here is to employ two types of zero-knowledge proof *techniques* — one for each type of operation.

[3]For instance, a recent work [BFH+20] used Ligero++ to produce a proof for linear model. Even for such a simple training task with a data set of size 2000, it ran out of memory (See Section 1.3).

[4]Arithmetic circuits only have addition and multiplication gates with respect to the underlying field.

[5]Many of the existing private preserving machine learning (PPML) literature considers training with fixed-point real numbers. See, for instance, [LJLA17, CPR18, AS19, RRG+21, KVH+21].

Assuming each of these techniques are well-suited for the respective type of operations, we propose a novel way to compose the two proof techniques to obtain an efficient zero-knowledge proof system for C.

Traditionally, zkSNARKs are not streaming friendly and require memory proportional to the entire circuit[6], whereas MPCitH is streaming friendly and only requires memory proportional to what is needed when simply evaluating the circuit. Our new framework preserves streaming friendliness by carefully combining techniques from the succint proofs literature with MPCitH. We refer the reader to Section 7 for a more detailed discussion on why our protocol is streaming friendly.

At a high-level, our proof will consist of commitments to a number of intermediate states in the circuit execution. Furthermore, we generate a proof for each type of operation, proving that the execution of that operation is consistent with the committed states and the output. Evidently, the proof size and the prover time here will be a sum of that in each proof system, which depends on the number/size of operations of each type.

In this work, we apply this framework to the training task. Naturally, we divide the operations into arithmetic and non-arithmetic operations. We use a custom MPC-in-the-head style zero-knowledge proof for the non-arithmetic operations and techniques from the zkSNARK literature for the arithmetic operations.

Main technical novelty. In MPC-in-the-head style protocols, the prover executes an MPC protocol for the circuit C and commits to the view of each party in the MPC. The verifier picks a random subset of parties and asks the prover to reveal the views of those parties in the clear. The verifier accepts the proof if the revealed views of the parties are consistent with an honest execution of the MPC protocol. The zero-knowledge property follows from the privacy of the MPC protocol.

As discussed earlier — in comparison to zkSNARKs — while this technique is better suited to handle real number operations, the proof size here is linear in the circuit size. To overcome this limitation, we make use of the techniques from zkSNARKs. In particular, when the verifier requests the opening of a party's view, we do not send the entire view in the clear. Instead, we give a succinct proof proving that the arithmetic operation each party does locally is done correctly. Note that some parts of the view are still revealed in the clear to allow the verifier to verify that the non-arithmetic operations are done correctly.

Custom MPC: The above use of zk-SNARKs helped us partially reduce the proof size by allowing the prover to not reveal the entire view of each party. However, the communication of the MPC protocol is still entirely revealed, which itself could be large (and contribute to a large proof size). Recently, there has been a lot of progress [BGJK21, GPS22, EGPS22] on reducing the communication complexity of MPC protocols using techniques of packed secret sharing [FY92]. We design a custom MPC protocol for logistic regression based on packed secret sharing to further reduce the communication complexity of our MPC protocol (and have it be sublinear in the size of the computation). This, in turn, helps us ensure that the proof size is sublinear in the size of the data and also improves the prover run-time.

It is worth noting that effectively implementing the MPC-in-the-head template typically requires starting with an MPC protocol that includes "some notion of robustness" against malicious adversaries. As discussed earlier, ML training is usually done over real-valued data. We use fixed-point arithmetic to represent real numbers with a fixed decimal precision using field elements[7] and borrow techniques from [CS10] to design our MPC protocol that works for fixed-point arithmetic. When working with this representation, we need to include a *truncation step* (which is a non-arithmetic operation) after every multiplication gate in the circuit. The way to handle these truncation steps (and all other non-arithmetic operations in ML

[6]Some exceptions include recent works such as [BCHO22, BHR+20, BHR+21] that introduce new techniques to design memory-friendly zkSNARKs at the cost of increasing the proof size.

[7]we refer the reader to Section 2.4 for a detailed discussion on how fixed-point arithmetic works

training) inside an MPC is to have the parties collectively sample shares of random field elements from a certain range. Sampling them in a distributed manner within an MPC requires adding additional checks to ensure they are sampled from the right distribution (necessary for robustness), which adds significant overhead to the MPC protocol. We avoid this overhead by designing an MPC in the *preprocessing hybrid model*, where we assume that these shares were generated in the preprocessing phase by a "trusted entity". In our MPC-in-the-head protocol, we allow the prover to act as this trusted entity to generate these random shares and give zkSNARK proofs[8] to the verifier certifying their validity and consistency. We defer more details to the technical sections of the paper.

Summary. In summary, our approach naturally enjoys the best of both worlds by combining MPC-in-the-head-based ZK proofs with zkSNARKs. That is, it enjoys a low prover time for the non-arithmetic operations and a succinct proof size for the arithmetic operations. As a result, our proof is both concretely efficient and of size sublinear in the data set.

Training Function. As a very first step toward designing concretely efficient proof systems for honest machine learning, we consider training linear models with logistic regressions in this work. Given a set of data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathbb{R}^D \times \{0,1\}$, the objective is to train a good linear model $\mathbf{w} \in \mathbb{R}^D$ under some activation function $f : \mathbb{R} \to (0,1)$ and the cross-entropy loss function.[9] We consider the stochastic gradient descent method with batching [MR18] for the training. In this setting, the training algorithm starts with some initial weights $\mathbf{w}_0 \in \mathbb{R}^D$ and iteratively updates the model $\mathbf{w}$ as

$$\mathbf{w}_j \leftarrow \mathbf{w}_{j-1} - \frac{a}{B} \cdot \mathbf{X}_j^\intercal \times \left(f\left(\mathbf{X}_j \times \mathbf{w}_{j-1}\right) - \mathbf{y}_j\right),$$

where a is some constant and $(\mathbf{X}_j, y_j)$ are different B-sized batches of the data. We will sometimes use the term models and weights interchangeably. We refer the reader to Section 2.2 for more details.

Limitations and Extensions of Our Techniques. One limitation of our techniques is that we work with a slightly modified training algorithm. For instance, we used a piece-wise linear function as an approximation of the activation function in logistic regression. Moreover, we employ fixed-point arithmetic instead of floating-point arithmetic. These slight modifications are fairly standard in privacy-preserving ML and have been shown to not greatly affect the accuracy [CBD15, MZ17, CDNF+21, EMMZ20]. We should note that, if one insists on proving that the training was done correctly without these simplifications, then the efficiency of the prover will likely worsen.

However, if one is willing to make these slight modifications to the training procedure, then the techniques that we develop in this paper could be *more generally beneficial* for designing concretely efficient zero-knowledge proof systems, that achieve an *appropriate trade-off* between proof generation time and proof size. For instance, there are no fundamental barriers to applying our techniques for designing a zkPoT for more complicated models such as convolution neural networks (CNN) with fixed-point arithmetic and where the activation function is approximated with appropriate piece-wise linear functions and all other operations are essentially matrix multiplications (we discuss more details in Section 6.3). However, one will need to work out the details and further optimize this protocol, since the training time in such models without any privacy/integrity guarantees is already significantly longer. We leave these extensions as exciting directions for future work.

Finally, while our zkPoT achieves sublinear proof size and verification time, they are not of constant size. In particular, they are quite larger than zkSNARK proofs. However, this is a price we pay for a reasonable proof generation time.

[8]For this, we design custom zk-SNARK proofs for some specific relations.

[9]That is, $H(a,b) = -a \cdot \log b - (1-a) \cdot \log(1-b)$ for all $a, b \in (0,1)$.

1.3 Related Works

Recently, many works [LKKO20, LXZ21a, FQZ$^+$21, blo22a, blo22b] have explored using zk-SNARKs for secure inference. Here, the objective is to convince the user of the integrity of the machine learning predictions. In general, inference involves a much smaller computation compared to training and, hence, the use of zk-SNARKs has found success in this setting.

To our best knowledge, Ligero++[BFH$^+$20] is the only work that mentions that their succinct proof system can be used for training. They provide estimates when trying to prove that linear regression was done correctly. Compared to our work, linear regression is a much simpler (and less commonly used) model than logistic regression, as there are no non-linear activation functions. Even for this simple task, Ligero++ runs out of memory after a training dataset of 2000, even when using a machine with 64 GB of RAM. More crucially, their proof system *does not* prove that the model is *trained* honestly; rather, it only proves that the final model has a high *accuracy* under the mean square loss function by utilizing the closed-form solution of linear regression. This is highly limited and undesirable for many reasons. First, it does not generalize to more complicated models, such as logistic regression, which does not have a closed-form solution. Second, training in practice does not always produce the optimal solution under the corresponding loss function, which renders this approach infeasible. Finally, for many training tasks, it is critical not only to achieve high accuracy, but also to ensure that the training process follows specific algorithms to satisfy fairness, differential privacy, and other requirements [CDG18, PMSW18].

In the SNARKs literature, several works [SVP$^+$12, WYX$^+$21, GJJZ22] also initiated the study of improving the efficiency of SNARKs for non-arithmetic operations such as floating-point numbers and integers.

On the other hand, designing efficient MPC protocol for machine learning has been extensively studied in the private preserving machine learning (PPML) literature. By now, we have a wide range of efficient protocols [MZ17, MR18, CGR$^+$19, WGC19, ASKG19, CRS20, KVH$^+$21, PSSY21, RBS$^+$22] computing various ML tasks. Motivated by practical scenarios, those works typically consider protocols among a small number of parties, for instance, two parties or three parties. The MPC-in-the-head framework that we consider requires a large number of parties, for instance, $n = 128$. This is because the soundness of the MPC-in-the-head approach grows exponentially in the number of parties.

Proof of Learning. The work of [JYCC$^+$21] introduces the notion of proof of learning that is closely related to the notion of proof of training. According to [JYCC$^+$21], a proof of learning protocol "demonstrates that a party has expended the computation required to obtain a set of model parameters correctly" and that "an adversary seeking to illegitimately manufacture a proof-of-learning needs to perform at least as much work than is needed for gradient descent itself". The key difference between our work and proof of learning is that we do not care about the running time the adversary needs to create a proof (except for running in polynomial time). The adversary should not be able to create a proof unless they perform the underlying computation. Additionally, a proof of learning does not care about the zero-knowledge which is a key aspect of our study. The notion of proof of learning has been subsequently studied in several works[TJSP22, Yam22, MWW22, FJT$^+$22]. Notably, the work of [KCC23] shows that if one uses the proof of learning protocols introduced in [JYCC$^+$21] as a proof for correctness, then they can indeed forge these proofs and show that models are trained privately.

1.4 Road Map

The rest of the paper is organized as follows. Section 2 includes the necessary preliminaries. Section 3 formally presents our proof of training model. Section 4 gives the MPC protocol for logistic regression. Section 5 introduces several sub-routines for giving a succinct proof of specific relations. Section 6 presents our final scheme, which combines the MPC protocol in Section 4 with the sub-routines in Section 5. Finally, Section 7 discusses the concrete efficiency of our scheme.

2 Preliminaries

In this work, we use λ as the computational security parameter. We use $\mathsf{negl}(\lambda)$ for a negligible function, which means that for all polynomial $f(\lambda)$, $\mathsf{negl}(\lambda) < 1/f(\lambda)$ for large enough λ. In addition, we use κ for the statistical security parameter. Next, we recall some useful primitives for our construction.

2.1 (Packed) Secret Sharing

Threshold secret-sharing schemes allow one to distribute a secret $s \in \mathbb{F}$ among n parties such that any $t+1$ parties can reconstruct the secret, while any t parties cannot learn any information about the secret. In this work, we will use Shamir's secret sharing [Sha79]. To share a secret s, one picks a random polynomial f with degree t such that $f(\alpha_0) = s$. The secret share of the i^{th} party is $f(\alpha_i) \in \mathbb{F}$, where $\alpha_0, \alpha_1, \ldots, \alpha_n$ are some fixed field elements that are FFT-friendly.

Shamir's secret sharing can be extended to share multiple secrets, known as packed secret sharing [FY92]. In particular, to pack ℓ secrets $(s_1, s_2, \ldots, s_\ell) \in \mathbb{F}^\ell$, one picks a degree t polynomial f such that $f(\alpha_0) = s_1, f(\alpha_{-1}) = s_2, \ldots, f(\alpha_{-(\ell-1)}) = s_\ell$. The i^{th} secret share shall be $f(\alpha_i) \in \mathbb{F}$. Again, $\alpha_{-(\ell-1)}, \ldots, \alpha_{-1}, \alpha_0, \alpha_1, \ldots, \alpha_n$ are some fixed field elements that are FFT-friendly. In this case, privacy will hold against a corruption of any $t_p = t - \ell$ parties, while $t+1$ parties are sufficient to reconstruct all ℓ secrets.

Well-Formedness of Secret Shares

In our protocol, we sometimes need to check that the secret shares are well-formed. Note that the secret shares are well-formed if and only if they form a Reed-Solomon codeword with dimension $t+1$ and block length n. Therefore, one may check the well-formedness by using the parity check matrix[10] $H \in \mathbb{F}^{n\times(n-t+1)}$. That is, $(s_1, \ldots, s_n)$ is well-formed if and only if $(s_1, \ldots, s_n) \cdot H = \mathbf{0}$. In particular, one may pick a random vector $r \in \mathbb{F}^{n-t+1}$, which gives a random vector $H \cdot r^\intercal$ in the dual space of the Reed-Solomon code, and check if $(s_1, \ldots, s_n) \cdot (H \cdot r^\intercal) \overset{?}{=} \mathbf{0}$. If this holds, with $1 - 1/|\mathbb{F}|$ probability, one can be convinced that $(s_1, \ldots, s_n)$ is well-formed.

Notations. Throughout this paper, we use the following notations. n stands for the number of parties. t stands for the threshold. ℓ represents the number of secrets each packed secret shares pack. For any long vector $\mathbf{X}$, we use $(\mathbf{X})^{(i)}$ to denote the i^{th} pack of ℓ elements. For example, if $\mathbf{X} = (x_1, x_2, \ldots)$, $(\mathbf{X})^{(2)}$ denotes $(x_{\ell+1}, \ldots, x_{2\ell})$. In the case that $\mathbf{X}$ is a matrix, we use $(\mathbf{X})_{\langle k,*\rangle}$ to denote the k^{th} row of the matrix $\mathbf{X}$. Similarly, $(\mathbf{X})^{(i)}_{\langle k,*\rangle}$ denotes the i^{th} chunk of the k^{th} row of the matrix $\mathbf{X}$.

For any ℓ secrets $\boldsymbol{\alpha}$, we use $[\![\boldsymbol{\alpha}]\!]$ to denote the packed secret sharing of α. For example, $\left[\!\!\left[(\mathbf{X})^{(i)}_{\langle k,*\rangle}\right]\!\!\right]$ to denote the packed secret sharing of $(\mathbf{X})^{(i)}_{\langle k,*\rangle}$. Finally, during the MPC protocol, the degree of the polynomial grows from t to $2t$ after each multiplication. To denote the packed secret shares that have degree $2t$, we will use the notation, for instance, $[\![(\mathbf{X})^{(i)}]\!]_{2t}$.

[10]The parity matrix can be efficiently computed. See, for example, Theorem 5.1.6 of [Hal15] for one explicit representation of the parity check matrix of the Reed-Solomon code.

2.2 Machine Learning

In this paper, we design efficient proofs for logistic regression. Similar to prior work [MR18], we consider the following specification of these learning tasks.

Logistic Regression. For a set of data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathbb{R}^D \times \{0, 1\}$, we want to find a linear model $\mathbf{w} \in \mathbb{R}^D$ such that $\mathbf{w} \times \mathbf{x}$ gives a "good" prediction of y. Given an *activation function* $f : \mathbb{R} \to (0, 1)$, the closeness is measured by the cross-entropy function

$$\frac{1}{N} \sum_{i \in [N]} \Big(- y_i \cdot \log f(\mathbf{w} \times \mathbf{x_i}) - (1 - y_i) \cdot \log f(1 - \mathbf{w} \times \mathbf{x_i}) \Big).$$

In logistic regression, the activation function is usually chosen as $f(x) = 1/(1 + e^x)$. We consider the stochastic gradient descent method with batching for training $\mathbf{w}$. In this method, the data points are randomly divided into batches of size B: $\mathbf{X}_1, \ldots, \mathbf{X}_{N/B}$. The initial weight $\mathbf{w}_0$ is set to be an arbitrary vector. Next, one applies the batches of data iteratively to update the weight vector

$$\mathbf{w}_j \leftarrow \mathbf{w}_{j-1} - \frac{a}{B} \cdot \mathbf{X}_j^{\intercal} \times (f\left(\mathbf{X}_j \times \mathbf{w}_{j-1}\right) - \mathbf{y}_j).$$

Here, a is some appropriate constant known as the learning rate. After exhausting all the batches, one may rerandomize the data points into another set of batches and repeat the process.

Since e^x is an expensive operation for MPC, we use the following approximation of f as the activation function (similar to [MR18])

$$f(x) = \begin{cases} 0 & x < -1/2 \\ x + 1/2 & -1/2 \leqslant x < 1/2 \\ 1 & 1/2 \leqslant x \end{cases}.$$

Regularization. In practice, it is important to train machine learning models with some form of regularization to prevent overfitting. For linear regression, the commonly used regularization function is L2 regularizer. That is, instead of minimizing the original loss function $\mathsf{Loss}(\mathbf{w}, \mathbf{X}, \mathbf{Y})$, one minimizes $\mathsf{Loss}(\mathbf{w}, \mathbf{X}, \mathbf{Y}) + c \cdot \sum_{i=1}^{D} w_i^2$ for some appropriate constant c. Applying the gradient descent method to this modified loss function yields an almost identical iterative updating function on $\mathbf{w}$. We omit this practical consideration in our construction but keep a note here that it can be added without any additional changes.

2.3 Secure Multiparty Computation

Secure Multiparty Computation (or MPC) is a distributed protocol that allows a group of n mutually distrusting parties to jointly compute a function on their private inputs, in a manner such that nothing beyond the output of the function is revealed. For the sake of completeness, we recall the definitions of t_p-privacy and t_r-robustness from [IKOS07] that will be crucially used when instantiating an MPC-in-the-head with a given MPC.

Definition 1 (t_p-Privacy [IKOS07]). *Let $1 \leqslant t_p < n$. We say that Π realizes f with computational t_p-privacy if there is a PPT simulator* sim *such that for any inputs $x_1, \ldots, x_n$ and every set of corrupt players $\mathcal{I} \subseteq [n]$ such that $|\mathcal{I}| \leqslant t_p$, the joint view $\mathsf{View}_{\mathcal{I}}(x_1, \ldots, x_n)$ of the players in $\mathcal{I}$ and $\mathsf{sim}(\mathcal{I}, \{x_i\}_{i \in \mathcal{I}}, f(x_1, \ldots, x_n))$ are (identically distributed, statistically close, computationally close).*

Definition 2 (Statistical t_r-Robustness [IKOS07])**.** *Let $1 \leqslant t_r < n$. We say that Π realizes f with statistical t_r-robustness if (1) it correct evaluates f in the presence of a semi-honest adversary (with at most negligible error) and (2) if for any computationally unbounded malicious adversary corrupting a set $\mathcal{I}$ of at most t_r players, and for any inputs $(x_1, \dots, x_n)$, if there is no $(x'_1, \dots, x'_n)$ such that $f(x'_1, \dots, x'_n) = 1$, then the probability that some uncorrupted player outputs 1 in an execution of Π in which the inputs of the honest player are consistent with $(x_1, \dots, x_n)$ is negligible in the security parameter.*

2.4 Secure Fix-point Arithmetic Operations

This section describes how to do the secure fix-point arithmetic computation over a prime finite field $\mathbb{F}_q$. Our approach follows the prior work of [CS10]. We consider fix-point real numbers with m bits precision, i.e., in the binary representation, there are m bits after the decimal point.

Representation of Real Number. For a real number a, we use $\mathsf{Rep}(a) \in \mathbb{F}_q$ to denote the field element that represents a, which is defined as $\mathsf{Rep}(a) = (a \cdot 2^m) \mod q$. Note that $a \cdot 2^m$ is an integer since we consider m bits precision. Moreover, given a representation $\mathsf{Rep}(a)$, we could always recover the real number a as

$$a = \begin{cases} \mathsf{Rep}(a)/2^m & \text{when } \mathsf{Rep}(a) < q/2 \\ (\mathsf{Rep}(a) - q)/2^m & \text{when } \mathsf{Rep}(a) > q/2 \end{cases}.$$

Here, the first case (resp., second case) corresponds to a being positive (resp., negative). In order to have correctness, we need the field size q to be large enough. In particular, we require that $q \geqslant 2^{M+m+\kappa}$. Here, M is an upper bound on the maximum number that we encounter during training, i.e., we assume $|a| \cdot 2^m$ will never exceed 2^M. The κ is the statistical security parameter. Looking ahead, our protocol for secure fix-point arithmetic operations will be $2^{-\Theta(\kappa)}$ statistically secure.

Secure Addition/Subtraction. Addition and subtraction can be simply done as follows. Observe that $\mathsf{Rep}(a + b) = \mathsf{Rep}(a) + \mathsf{Rep}(b)$. Here, the first addition is over the real number, and the second addition is over the field. Similarly, $\mathsf{Rep}(a - b) = \mathsf{Rep}(a) - \mathsf{Rep}(b)$. Based on this, parties could perform addition and subtraction non-interactively. For instance, if parties hold secret sharing $[\![\mathsf{Rep}(a)]\!]$ and $[\![\mathsf{Rep}(b)]\!]$, they may simply locally add their secret shares to get shares of $\mathsf{Rep}(a + b)$, i.e., $[\![\mathsf{Rep}(a)]\!] + [\![\mathsf{Rep}(a)]\!] = [\![\mathsf{Rep}(a + b)]\!]$.

Secure Multiplication. For multiplication, observe that $\mathsf{Rep}(a) \cdot \mathsf{Rep}(b)$ does not represent $a \cdot b$ but $a \cdot b \cdot 2^m$ instead. Therefore, we need to truncate the last m bits of $\mathsf{Rep}(a) \cdot \mathsf{Rep}(b)$ to restore correctness. The following protocol [CS10] is a secure truncation protocol with $2^{-\kappa}$ statistical security.

Secure Truncation. This protocol truncates a number a by m bits. Let $\alpha = \mathsf{Rep}(a)$. Parties hold secret shares of two masks γ_{high} and γ_{low}, which are sampled randomly from $\{0, \dots, 2^{M+\kappa}\}$ and $\{0, \dots, 2^m\}$, respectively. Let $\gamma = \gamma_{\mathsf{high}} \cdot 2^m + \gamma_{\mathsf{low}}$.

1. $[\![\beta]\!] = [\![\alpha]\!] + 2^{M+m}$.
2. Parties reconstruct $\gamma' = \gamma + \beta$ by broadcasting their secret shares $[\![\gamma + \beta]\!]$.
3. Parties compute $([\![\alpha]\!] + [\![\gamma_{\mathsf{low}}]\!] - (\gamma' \mod 2^m)) \cdot 2^{-m}$ locally as the share of the truncated value.

The first step is to ensure that the β represents a positive value. The γ_{low} is to mask the last m bits. The γ_{hish} is to mask the high order bits. Since we are masking integers, we need γ_{hish} to be uniform over a range that is 2^κ times bigger than the maximum integer we are masking. This ensures that the protocol is $2^{-\kappa}$ insecure. Note that this is a probabilistic truncation protocol.[11]

In our actual protocol, we use this protocol with packed secret sharing. The difference is that the reconstructed secret γ' is a vector of ℓ secret. Therefore, parties do not subtract $\gamma' \mod 2^m$ in the last step, but subtract the respective packed secret share of $[\![\gamma' \mod 2^m]\!]$.

Secure "Less-than". Finally, we note that the secure truncation protocol can be used to compute the "less than" function securely. Here, given the input x, we want to compute $\mathbb{1}_{x>c} = \begin{cases} 1 & x > c \\ 0 & x \leqslant c \end{cases}$ for some constant c. Suppose $x - c \in \{-2^M, 2^M\}$. Then $\mathbb{1}_{x<c}$ is exactly the $(M+1)^{th}$ order bits of $x - c + 2^M$. Therefore, we could use a secure truncation protocol to securely compute $\mathbb{1}_{x>c}$ for any constant c.

One subtle issue is that the truncation protocol we present earlier is for probabilistic truncation. Here, we want to compute the floor function $\left\lfloor \frac{x-c+2^M}{2^M} \right\rfloor$. In the MPC-in-the-head setting, the trick is that the prover will always pick the mask for the lower-order bits γ_{low} appropriately such that the last M bits of the masked value are all 1. This ensures: (1) Correctness – there is no overflow for the lower-order bits, which means that we are doing the floor function instead of probabilistic truncation; (2) Soundness – all the masked bits are one guarantees zero-knowledge.

2.5 (List) Polynomial Commitment

Polynomial commitment scheme allows the committer C to use a short commitment to commit to a polynomial. Later, it can open this polynomial at any location and convince the receiver $\mathcal{R}$ that the opening is consistent with the commitment. In this work, we use list polynomial commitment [KPV22], which is a relaxed variant of polynomial commitment. It has the following components.

- $\sigma \leftarrow \mathsf{Com}(f)$: For any polynomial f of degree $\leqslant d$, $\mathsf{Com}(f)$ computes a succinct commitment of f.
- $b \leftarrow \langle C(f, \sigma, \{(x_i\}_i), \mathcal{R}(\sigma, \{(x_i, y_i\}_i) \rangle$: is an interactive protocol between the committer C and the receiver $\mathcal{R}$. The committer has input $f, \sigma, \{x_i\}_i$ and the receiver has input $\sigma, \{(x_i, y_i\}_i$. This protocol allows C to convince $\mathcal{R}$ that σ is a commitment of some polynomial f such that $y_i = f(x_i)$ for all i. At the end of the protocol, the receiver $\mathcal{R}$ outputs either $b = 0$ or $b = 1$.

A polynomial commitment should satisfy both correctness and binding properties. For correctness, the verifier should always accept honestly generated proofs. Formally, for all polynomial f of degree $\leqslant d$ and a set of evaluation points $\{x_i\}_i$, it holds that

$$\Pr\left[b = 1 \middle| \begin{array}{c} \sigma \leftarrow \mathsf{Com}(f) \\ b \leftarrow \langle C(f, \sigma, \{x\}_i), \mathcal{R}(\sigma, \{x_i, f(x_i)\}_i) \rangle \end{array} \right] = 1.$$

The binding property for the t-list polynomial commitment requires that, for any commitment $\sigma \leftarrow \mathcal{A}(1^\lambda)$ generated by a malicious committer $\mathcal{A}$, there exists a list of degree $\leqslant d$ polynomials $L = \{f_1, f_2, \ldots, f_t\}$ such that, for any set of points $\{x_i, y_i\}_i$, we have

$$\Pr\left[\begin{array}{c} \exists f \in L \text{ s.t.} \\ \forall i,\ f(x_i) = y_i \end{array} \middle| 1 \leftarrow \langle \mathcal{A}(f, \sigma, \{x\}_i), \mathcal{R}(\sigma, \{x_i, f(x_i)\}_i) \rangle \right] = 1 - \mathsf{negl}(\lambda).$$

[11]For instance, to truncate 1.1 to be an integer, it will output 2 with probability 0.1 and 1 with probability 0.9.

There are many polynomial commitment schemes in the literature. In this work, we use the polynomial commitment scheme based on FRI [BBHR18]. This scheme avoids the use of "expensive" group operations and relies only on "cheap" cryptographic tools such as cryptographic hash functions, which is crucial for concrete efficiency. Furthermore, the relaxation from polynomial commitment to list polynomial commitment further help reduce the prover's cost and proof size for FRI-based polynomial commitment schemes. Overall, it has the following properties.

- Constant commitment size: $|\sigma| = O(1)$.
- The interactive protocol $\langle C, \mathcal{R} \rangle$ has logarithmic (i.e., $O(\log d)$) rounds and $O(\log^2 d)$ overall communication.

By invoking Fiat-Shamir heuristics, it gives a non-interactive opening proof with size $O(\log^2 d)$. We refer the readers to [Sta21] for more details. In particular, we employ the "grinding" techniques (refer to Section 3.11.3 of [Sta21]) to further reduce the concrete proof size.

Privacy. Our protocol employs the polynomial commitment schemes through the following paradigm. Initially, the message is encoded using randomized (Reed-Solomon) encodings, after which the encodings are committed using the FRI-based polynomial commitment schemes. The final proof will give a few opening proofs of the committed polynomials at a random location. These opening proofs do not leak anything since the randomized encoding ensures that the symbols at any small number of locations reveal no information about the encoded message. These points are discussed within the proof when we argue the zero-knowledge property.

2.6 Protocols for Sumcheck and Permutation

In this section, we present two protocols that we employ as a subroutine in our final construction. These protocols allow the prover to prove useful relations about the committed values.

Sumcheck protocol. The first protocol is a univariate sumcheck protocol, introduced by [BCR+19]. Let $\mathbb{Q} = \{\omega, \omega^2, \ldots, \omega^{\mathsf{s}} = 1\}$ be a multiplicative subgroup of $\mathbb{F}$. Let $f(x)$ be a polynomial such that $f(\omega), f(\omega^2), \ldots, f(\omega^{\mathsf{s}})$ encodes the information that we are interested in. The sumcheck protocol lets the prover give a succinct proof proving the sum of the encoded values $\sum_{i=1}^{\mathsf{s}} f(\omega^i)$. In particular, we have the following lemma and univariate sumcheck protocol.

Lemma 1. *Given a polynomial $f(x)$ of arbitrary degree, the sum of the evaluation of $f(x)$ on the set $\mathbb{Q} = \{\omega, \omega^2, \ldots, \omega^{\mathsf{s}} = 1\}$ is γ if and only if there exist two quotient polynomials $Q_1(x)$ and $Q_2(x)$ such that (1) the degree of $Q_1(x)$ is at most $\mathsf{s} - 2$ and (2) the following polynomial identity holds.*

$$f(x) = \gamma/\mathsf{s} + Q_1(x) \cdot x + Q_2(x) \cdot Z(x).$$

Here, $Z(x)$ is the vanishing polynomial for $\mathbb{Q}$ defined as $Z(x) = \prod_{i=1}^{\mathsf{s}}(x - \omega^i) = x^{\mathsf{s}} - 1$.

Statement. The prover and verifier hold a commitment of some polynomial $p(x)$ of degree n. The prover wants to prove that $\sum_{i=1}^{\mathsf{s}} p(\omega^i) = \alpha$.
Protocol Description.

1. The prover samples and commits to a polynomial $q(x)$ of the same degree as $p(x)$. In addition, it sends $\beta = \sum_{i=1}^{\mathsf{s}} q(\omega^i)$.

2. The verifier sends random challenges c.

3. The prover computes the quotient polynomial $Q_1(x)$ and $Q_2(x)$ according to Equation 1 with $f(x) = c \cdot p(x) + q(x)$ and $\gamma = c \cdot \alpha + \beta$. The prover sends the commitment of $Q_1(x)$ and $Q_2(x)$ to the verifier.

4. The verifier picks a random point r.

5. Prover sends the evaluation of all the polynomials at r: $p(r), q(r), Q_1(r), Q_2(r)$.

6. Prover and receiver invoke the polynomial commitment protocol to prove

   - All the evaluations at r are correct.
   - The degree of $Q_1(x)$ is $\leqslant \mathsf{s} - 2$.

7. Verifier accepts the proof if (1) the verification in the last step succeeds; (2) Equation 1 holds at $x = r$.

Figure 1: Sum-Check Protocol

Lemma 2 ([BCR+19])**.** *The univariate sumcheck protocol is correct and sound. That is, (1) the verifier will accept the proof with probability 1 when interacting with an honest prover. (2) the verifier will reject the proof with probability* $1 - \mathsf{negl}(\lambda)$ *for any false statement that a malicious prover tries to prove.*

Remark. In the protocol, we often need to use the sumcheck protocol on the product or sum of polynomials, e.g., $f(x) = f_1(x) + f_2(x)$ or $f(x) = f_1(x) \cdot f_2(x)$, where $f_1(x)$ and $f_2(x)$ are committed. Note that, one may still run the sum-check protocol above. In cases where one needs to open $f(x) = f_1(x) + f_2(x)$ at a random location $x = r$, one may simply open $f_1(x)$ and $f_2(x)$ at $x = r$ separately, and derive $f(r) = f_1(r) + f_2(r)$.

On another note, we also sometimes need to do sumcheck on the difference between two polynomials $f_1(x)$ and $f_2(x)$, where they encode values on different subgroups. For instance, $f_1(x)$ encodes information on $\mathbb{Q} = \{\omega, \omega^2, \ldots, \omega^{\mathsf{s}} = 1\}$, while $f_2(x)$ encodes information on $\mathbb{H} = \{\mu, \mu^2, \ldots, \mu^{\mathsf{s}/t} = 1\}$. In such cases, we may do sumcheck on the polynomial $f_1(x) - f_2(x) \cdot I(x)$, where $I(x) = 1$ for $x \in \mathbb{H}$ and $I(x) = 0$ for $x \in \mathbb{Q} \setminus \mathbb{H}$.

Permutation Check Protocol. The second protocol checks whether a sequence of committed values satisfies a specific permutation relation. This protocol is due to Plonk [GWC19]. Looking ahead, this protocol is useful when we want to check the data used during training across different iterations are correctly permutated.

Let $\mathbb{Q} = \{\omega, \omega^2, \ldots, \omega^{\mathsf{s}} = 1\}$ be a multiplicative subgroup.Let $f_1(x), \ldots, f_k(x)$ and $g_1(x), \ldots, g_k(x)$ be polynomials such that their evaluations on $\mathbb{Q}$ encodes the information that we are interested in. Let $\mathsf{Perm} : [k \cdot \mathsf{s}] \to [k \cdot \mathsf{s}]$ be a permutation. The permutation check protocol allows the prover to give a succinct proof proving that the vectors

$$\boldsymbol{f} = (f_1(\omega), \ldots, f_1(\omega^{\mathsf{s}}), \ldots, f_k(\omega), \ldots, f_k(\omega^{\mathsf{s}}))$$
$$\boldsymbol{g} = (g_1(\omega), \ldots, g_1(\omega^{\mathsf{s}}), \ldots, g_k(\omega), \ldots, g_k(\omega^{\mathsf{s}}))$$

are consistent with Perm, i.e., $\boldsymbol{f}_i = \boldsymbol{g}_{\mathsf{Perm}(i)}$ for all i. For this protocol, we define the following degree

$\leqslant \mathsf{s}-1$ polynomials $I_1, I_2, \ldots, I_k$ and $J_1, J_2, \ldots, J_k$ based on Perm. For all $i \in [k], j \in [\mathsf{s}]$, it satisfies

$$I_i(\omega^j) = (i-1)\cdot \mathsf{s} + j$$
$$J_i(\omega^j) = \mathsf{Perm}((i-1)\cdot \mathsf{s} + j)$$

Statement. The prover and verifier hold commitments to $f_1, f_2, \ldots, f_k$ and $g_1, g_2, \ldots, g_k$. The prover wants to prove that they satisfy the permutation defined by Perm.
Protocol Description.

1. The verifier sends random challenges β, γ.
2. The prover computes the following

$$f_i'(x) = f_i(x) + \beta \cdot I_i(x) + \gamma$$
$$g_i'(x) = g_i(x) + \beta \cdot J_i(x) + \gamma$$

   Let $f'(x) = \prod_{i=1}^k f_i'(x)$ and $g'(x) = \prod_{i=1}^k g_i'(x)$. The prover computes a polynomial $P(x)$ of degree $\leqslant \mathsf{s}-1$ defined as

$$P(\omega^i) = \prod_{1\leqslant j<i} f'(\omega^j)/g'(\omega^j)$$

   for all $i \in \{2, \ldots, \mathsf{s}\}$ and $P(\omega) = 1$. It also computes $Q_1(x)$ and $Q_2(x)$ according to Equation 1.

   It commits to $P(x), Q_1(x), Q_2(x)$ and sends the commitments to the verifier.
3. The verifier sends a random challenge r.
4. The prover sends the following $P(r), P(r\cdot\omega), Q_1(x), Q_2(x), f_1(r), \ldots, f_k(r)$, and $g_1(r), \ldots, g_k(r)$.
5. The prover and receiver invoke the polynomial commitment protocol to verify all the evaluations.
6. Verifier accepts the proof if (1) the verification in the last step succeeds; (2) the following polynomial identities hold at $x = r$.

$$\left\{\begin{array}{c} \frac{x^n-1}{x-\omega}\cdot (P(x)-1) = Z(x)\cdot Q_1(x) \\ P(x)\cdot f'(x) - P(x\cdot\omega)\cdot g'(x) = Z(x)\cdot Q_2(x) \end{array}\right\} \tag{1}$$

Figure 2: Permutation Check Protocol

Lemma 3 ([GWC19]). *The permutation check protocol above is correct and sound. That is, (1) the verifier will accept the proof with probability 1 when interacting with an honest prover. (2) the verifier will reject the proof with probability* $1-\mathsf{negl}(\lambda)$ *for any false statement that a malicious prover tries to prove.*

3 Formalizing Proof of Training

In this section, we formally present our notion of zero-knowledge proof of training as a cryptographic primitive. Consider a *public* learning algorithm C, which takes data and randomness as input, and outputs a model

$$\mathsf{model} = C(\mathsf{data}, \mathsf{rand}).$$

A proof of training for this training task consists of the following.

- $\sigma = \mathsf{Commit}(\mathsf{data}, \mathsf{rand})$: on input the data and randomness, outputs a commitment.
- $(\mathsf{model}, \sigma', \pi) = \mathsf{Prove}(\mathsf{data}, \mathsf{rand})$: on input the data and the randomness used in the training process, output the model, a commitment to the model σ', and a proof π.
- $b = \mathsf{Verify}(\sigma, \sigma', \pi)$: on input the commitment σ to the data and the randomness, the commitment σ' to the model, and a proof π, it output $b \in \{0, 1\}$.

The proof of training should satisfy the following guarantees.

- **Correctness.** An honestly trained model and its corresponding generated proof should always be accepted. That is, for any commitment $\sigma = \mathsf{Commit}(\mathsf{data}, \mathsf{rand})$, we have

$$\Pr\left[b = 1 \;\middle|\; \begin{array}{c} (\mathsf{model}, \sigma', \pi) = \mathsf{Prove}(\mathsf{data}, \mathsf{rand}) \\ b = \mathsf{Verify}(\sigma, \sigma', \pi) \end{array} \right] = 1$$

- **Soundness.** The prover cannot convince the verifier to accept an *incorrect* model. In particular, for any commitment $\sigma = \mathsf{Commit}(\mathsf{data}, \mathsf{rand})$ and PPT adversary $\mathcal{A}$, we have

$$\Pr\left[\begin{array}{c} b = 1 \wedge \\ \mathsf{model}^* \neq \mathsf{model} \end{array} \;\middle|\; \begin{array}{c} (\mathsf{model}, \sigma', \pi) = \mathsf{Prove}(\mathsf{data}, \mathsf{rand}) \\ (\mathsf{model}^*, (\sigma')^*, \pi^*) \leftarrow \mathcal{A}(\mathsf{data}, \mathsf{rand}) \\ b = \mathsf{Verify}(\sigma, (\sigma')^*, \pi^*) \end{array} \right] = \mathsf{negl}(\lambda).$$

- **Zero-knowledge.** The proof π should not reveal anything about the model. That is, there exists a simulator Sim such that, for any data and rand,

$$\mathsf{Sim}(1^\lambda) \approx \left\{ \begin{array}{c} \sigma = \mathsf{Commit}(\mathsf{data}, \mathsf{rand}) \\ (\mathsf{model}, \sigma', \pi) = \mathsf{Prove}(\mathsf{data}, \mathsf{rand}) \\ \text{Output } \sigma, \sigma', \pi \end{array} \right\}.$$

For the practicality of this primitive, we also have the following efficiency requirements. Let T be the running time of the algorithm.

- The proof is sublinear in T, i.e., $|\pi| = o(T)$.
- The prover's and verifier's complexity should be concretely efficient. Asymptotically, we require that the prover's complexity is $O(T \log T)$, while the verification complexity is $O(T)$.

For perspectives on the efficiency requirement, the existing SNARK approaches achieve give extremely short proofs $|\pi| = \mathrm{polylog} T$, but the prover's concrete running time is simply astronomical.[12] The MPC-in-the-head style proof, on the other hand, has great concrete efficiency, but the proof size is even larger than T, i.e., $|\pi| \geqslant T$.

A few remarks regarding our definitions are in order.

[12]For instance, for a training task that requires a few hours to train, the prover's running time will be a few days [blo22a].

- **Privacy of the model.** In the above definition, we consider the setting where the model is private. One may consider an analogous definition where the model is in the clear.
- **On use of commitments.** In our definition, by default, the common input consists of commitments to the dataset, model, and the randomness. When it comes to the randomness, it can even be generated by the verifier itself to prevent potential choices of malicious randomness selection [GKVZ22], and our protocols can directly adapt to this setting. Regarding using commitments to exchange the model and the data set, we note that doing so is a natural choice for the following reasons. First, we want the transcript uniquely determine the model and the data set (in fact, the data set can be seen as a common input given to the two parties). This means that the exchanged message should bind the prover to the model and the data set. Second, we want the whole protocol to remain zero-knowledge, which implies the exchanged messages should still hide the model and the data set.

Fine-Tuning over Foundational Model. In our definition, we assume that the commitment to the data and randomness is honestly generated by the trainer. For applying our definitions to proving fine-tuned models on foundation models, one may desire the assurance that the input data are indeed given by the feature extractor (i.e., foundation model). There are many generic ways one could ensure this. For instance, if the committed input is generated by the foundation model, one may assume that the foundation model also signs the commitment using a digital signature [GMR88]. Then, one may simply verify the digital signature to be convinced that the committed input is indeed produced by the foundational model. Or, if one considers the setting where the commitment is generated by the prover itself, one may assume that the foundation model signs the features that it outputs. Additionally, we request the prover to not only prove that the model is trained honestly, but also prove that the input is *well-formed*, which is to prove the fact that it knows valid signatures of the committed input.

These additional measures are relatively much cheaper than proving the honesty of training. Therefore, in this work, we omit these additional details and focus on the part of proving honest training.

4 MPC Protocol for MPC-in-the-Head

As discussed in the introduction, we design an MPC-in-the-head-based zero-knowledge proof of training for logistic regression, that is interspersed with techniques used in the design of zkSNARKs. In this section, we describe the MPC protocol that we use to instantiate our modified MPC-in-the-head framework. We start by giving a more detailed overview of our approach to highlight the choices involved in designing this MPC protocol.

4.1 Overview

Let us first recall the basic blueprint of how a zero-knowledge proof based on MPC-in-the-head works.

MPC-in-the-head: MPC-in-the-head [IKOS07] is a technique used for designing three-round, public-coin, zero-knowledge proofs of knowledge using MPC protocols (also called Σ-protocols). At a high level, the prover simulates an n-party MPC protocol Π virtually on the relation circuit $\mathcal{R}(x, \cdot)$, that has the statement hard-wired in it. The input to this MPC is the witness w (which is shared among the virtual parties) corresponding to the statement x, such that $\mathcal{R}(x, w) = 1$. In the first round of the protocol, the prover commits to the views of all the parties in this MPC execution. In the second round, an honest verifier then selects a random subset of the views to be opened. In the third round, the prover opens to the views of the selected players. The verifier then verifies that those views are consistent with each other and with an honest execution, where the output of Π is 1.

Zero-knowledge property in such a protocol follows from the fact that the verifier only gets to open the views of a subset of the virtual parties (the size of this subset depends on the number of corrupt parties that the underlying MPC protocol Π can tolerate). From the privacy of Π, it follows that these views do not leak any information about the witness. The soundness of this protocol depends on the fraction of parties whose views the verifier is allowed to open. For instance, if the verifier is allowed to open corr out of n parties' views, the soundness error in the above protocol is expected to be roughly $1/\binom{n}{\mathsf{corr}}$. It is easy to see that if $1/\binom{n}{\mathsf{corr}}$ is negligibly small, a single iteration of the above protocol ensures enough soundness. However, if this is not the case, then we need to repeat the above protocol enough number of times (typically proportional to the security parameter) using fresh randomness to amplify soundness.

MPC-in-the-head using an Honest Majority MPC: Let us now analyze what impact the choice of n and corr have on the communication complexity of the resulting zero-knowledge protocol. We want to avoid the communication overhead (dependent on the security parameter) from having to repeat the protocol in order to get sufficient soundness. Hence, we somehow want to ensure that a single iteration gives us enough soundness. For this, we can choose $n = O(\lambda)$ and $\mathsf{corr} = n/c$, where λ is the security parameter and c is some constant. For these values of n and corr, $1/\binom{n}{\mathsf{corr}}$ is negligibly small.

However, since the verifier gets to open the views of corr number of parties, the communication in our ZKP now depends on $\mathsf{corr} = O(\lambda)$. If the view of each party is proportional to the size of the relation circuit, overall the communication complexity will be $O(\lambda \cdot |\mathcal{R}|)$. This clearly does not seem to have helped us in avoiding additional dependence on the security parameter.

Our idea to avoid this is to instead use a special MPC protocol where the view of each party is somehow sublinear in the size of the related circuit. For instance, if the view of each party was $O(|\mathcal{R}|/n)$, then overall the communication complexity of the resulting ZKP will be $O(\mathsf{corr} \cdot |\mathcal{R}|/n) = O(|\mathcal{R}|)$. Such MPC protocols are known for special classes [BGJK21] of circuits using a technique known as *packed secret sharing* [FY92].

This, however, yields a zero-knowledge proof where the total communication is linear in the relation circuit (a.k.a., the data and logistic regression computation in our setting). As discussed in the introduction, training is typically done over massive data-sets, and it would be practically infeasible to communicate proofs that are linear in its size.

Optimizing Honest Majority MPC for Logistic Regression: Let us now discuss how we can hope to further improve the communication and make it sublinear in the size of the data. We observe that given the data $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$, and $y_1, \ldots, y_N \in \mathbb{R}$ (Note that this N here is different from the n that denotes the number of parties in our MPC protocol) and the initializing parameters, training a linear model primarily requires additions and inner-product computations. In honest-majority MPC protocols based on Shamir secret sharing or packed secret sharing, securely computing additions does not require any communication. Moreover, it is possible to securely compute inner products using communication independent of the length of inner-product vectors. This essentially allows us to design a packed secret sharing-based MPC protocol for this linear model (logistic regression) computation, where the total communication between the parties is $O(N)$ and is independent of D.

High-Level Description of Our MPC Protocol. Given the above background on some of our design choices, we now give a brief summary of our MPC protocoland then proceed to give a formal description in Section 4.2.

As discussed in Section 1.2, we use fixed-point arithmetic to represent real numbers as field elements and design an MPC protocol in the pre-processing model that works as follows:

- *Pre-Processing Phase:* We assume that a trusted entity implements this phase.
  1. *Data (and Model) Independent Preprocessing:* The trusted party generates some correlated randomness (independent of the data and weights), computes packed secret shares of these values and sends them to the parties.
  2. *Data Dependent Preprocessing:* The trusted party computes two different packed secret sharings of the data and sends them to the parties. Recall that, when using mini-batch gradient descent, the data $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$ is segregated into batches of the form $\mathbf{X}_1, \ldots, \mathbf{X}_{N/B} \in \mathbb{R}^{B \times D}$. The trusted party essentially computes two different packed secret sharings of each of these batches — one where the values in these matrices are packed and shared row-wise and another where the values are packed and shared column-wise. It then sends both types of shares to the parties.
- *Online Phase:* We assume that the inital weights vector $\mathbf{w}_0 \in \mathbb{R}^D$ is publicly known to all parties. Given this and all the above shares received in the pre-processing phase, the parties run a distributed algorithm to securely compute the logistic regression function and obtain packed secret shares of the resulting weights vector. The parties in this phase essentially perform fixed-point arithmetic computation (see Section 2.4 for reference) over packed secret shares.

Remark. We remark that the total computation and the total size of all the private states held by the parties in this MPC protocol are still $O(DN)$. Instantiating the MPC-in-the-head framework with this MPC protocol does not yet give us a zero-knowledge proof system, where communication is sublinear. Recall that in an MPC-in-the-head-based ZKP, when the verifier chooses to open the views of a subset of the parties, those views consist of the messages that the opened parties received from other virtual parties in the MPC as well as their entire private state. Here, in our MPC, since the total private state is still $O(DN)$ (i.e., $O(DN/n)$ per party), overall our communication will still be corr $\times O(DN/n) = O(DN)$. As discussed in the introduction, to avoid this overhead, we intersperse the verifier checks in our MPC-in-the-head protocol with techniques from the zkSNARKs literature. For this, we design a few zkSNARKs for specific relations which we discuss in Section 5.

4.2 Protocol Description

We now give a formal description of our n-party MPC protocol. We describe the pre-processing phase of our MPC in Section 4.2.1 and the online phase in Section 4.2.2.

4.2.1 Pre-Processing Phase.

$\forall j \in [N/B]$, the trusted entity proceeds as follows in the pre-processing phase:

- **Correlated Random Shares:**
  1. **Sampling Randomness:** Sample random vectors $\boldsymbol{\theta}_j, \boldsymbol{\mu}_j, \boldsymbol{\alpha}_j, \boldsymbol{\tau}_j \in \mathbb{F}^B$ and $\boldsymbol{\gamma}_j, \boldsymbol{\beta}_j \in \mathbb{F}^D$. For each $k \in [B]$, let $\boldsymbol{\nu}_{j,k}$ be an ℓ length vector containing ℓ copies of $\boldsymbol{\tau}_j[k]$ and for each $k \in [D]$, let $\boldsymbol{\delta}_{j,k}$ be an ℓ length vector containing ℓ copies of $\boldsymbol{\beta}_j[k]$.
  2. **Packed Shares:** Compute the following packed secret shares $\left\{ \left[\!\left[\left(\boldsymbol{\gamma}_j\right)^{(u)} \right]\!\right], \left[\!\left[\left(\boldsymbol{\gamma}_j\right)^{(u)} \right]\!\right]_{2t} \right\}_{u \in [D/\ell]}$, $\left\{ \left[\!\left[\boldsymbol{\nu}_{j,k} \right]\!\right] \right\}_{k \in [B]}$, $\left\{ \left[\!\left[\boldsymbol{\delta}_{j,k} \right]\!\right] \right\}_{k \in [D]}$, $\left\{ \left[\!\left[\left(\boldsymbol{\tau}_j\right)^{(u)} \right]\!\right]_{2t} \right\}_{u \in [B/\ell]}$, $\left\{ \left[\!\left[\left(\boldsymbol{\beta}_j\right)^{(u)} \right]\!\right]_{2t} \right\}_{u \in [D/\ell]}$,

$$\left\{\left[\!\left[(\boldsymbol{\alpha}_j)^{(u)}\right]\!\right], \left[\!\left[(\boldsymbol{\alpha}_j)^{(u)}\right]\!\right]_{2t}\right\}_{u\in[B/\ell]}, \left\{\left[\!\left[(\boldsymbol{\theta}_j)^{(u)}\right]\!\right]_{2t}, \left[\!\left[\left(\boldsymbol{\mu}_j\right)^{(u)}\right]\!\right]_{2t}, \left[\!\left[(\boldsymbol{\theta}_j)^{(u)}\right]\!\right], \left[\!\left[\left(\boldsymbol{\mu}_j\right)^{(u)}\right]\!\right]\right\}_{u\in[B/\ell]}.$$

- **Masks for Non-Arithmetic Computations:**
  1. **Sampling Masks:**
     - $\forall u \in [B/\ell]$, sample random vectors $\mathsf{mask}^{\downarrow}_{j,u} \in \{0, 2^m\}^\ell$, $\mathsf{mask}^{\uparrow}_{j,u} \in \{0, 2^{M+\kappa}\}^\ell$, $\mathsf{LT}^{\downarrow}_{j,u}, \overline{\mathsf{LT}}^{\downarrow}_{j,u} \in \{0, 2^M\}^\ell$ and $\mathsf{LT}^{\uparrow}_{j,u}, \overline{\mathsf{LT}}^{\uparrow}_{j,u} \in \{0, 2^{\kappa+1}\}^\ell$.
     - $\forall u \in [D/\ell]$, sample random vectors $\overline{\mathsf{mask}}^{\downarrow}_{j,u} \in \{0, 2^m\}^\ell$ and $\overline{\mathsf{mask}}^{\uparrow}_{j,u} \in \{0, 2^{M+k}\}^\ell$.
  2. **Packed Shares:** Compute packed secret shares $\left\{\left[\!\left[\mathsf{mask}^{\downarrow}_{j,u}\right]\!\right], \left[\!\left[\mathsf{mask}^{\uparrow}_{j,u}\right]\!\right]\right\}_{u\in[B/\ell]}$, $\left\{\left[\!\left[\mathsf{LT}^{\downarrow}_{j,u}\right]\!\right], \left[\!\left[\mathsf{LT}^{\uparrow}_{j,u}\right]\!\right]\right\}_{u\in[B/\ell]}$, $\left\{\left[\!\left[\overline{\mathsf{LT}}^{\downarrow}_{j,u}\right]\!\right], \left[\!\left[\overline{\mathsf{LT}}^{\uparrow}_{j,u}\right]\!\right]\right\}_{u\in[B/\ell]}$ and $\left\{\left[\!\left[\overline{\mathsf{mask}}^{\downarrow}_{j,u}\right]\!\right]. \left[\!\left[\overline{\mathsf{mask}}^{\uparrow}_{j,u}\right]\!\right]\right\}_{u\in[D/\ell]}$.
- **Data Shares:** Let the input training sample be $\mathbf{X}_j \in \mathbb{F}^{B\times D}$, $\forall j \in [N/B]$ and the corresponding output be $y_1, \ldots, y_N \in \mathbb{F}$. $\forall j \in [N/B]$, compute packed secret shares: $\left\{\left[\!\left[(\mathbf{y}_j)^{(u)}\right]\!\right], \left\{\left[\!\left[\left(\mathbf{X}_j^\intercal\right)^{(u)}_{\langle k,*\rangle}\right]\!\right]\right\}_{k\in[D]}\right\}_{u\in[B/\ell]}$ and $\left\{\left[\!\left[(\mathbf{X}_j)^{(u)}_{\langle k,*\rangle}\right]\!\right]\right\}_{k\in[B],u\in[D/\ell]}$.

  Finally, it sends the above shares to the respective parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$.

4.2.2 Online Phase.

Let $\mathbf{w}_0 \in \mathbb{F}^D$ be the initial weights vector. In addition to the shares received from the trusted party in the above pre-processing phase, we also assume that the parties have the following packed secret shares of the initial weights vector: $[\![\mathbf{w}_{0,1}]\!], \ldots, [\![\mathbf{w}_{0,D/\ell}]\!]$, where each $[\![\mathbf{w}_{0,u}]\!]$ is a packed secret sharing of an ℓ-length vector $(\mathbf{w}_{0,u}, \ldots, \mathbf{w}_{0,u})$ and $\mathbf{w}_{0,u}$ denotes the uth element in $\mathbf{w}_0$. Since the initial weights vector is public, we assume that these shares are deterministically computable given $\mathbf{w}_0$ and hence known to all parties. Given these shares, the parties proceed as described in Figure 3. Without loss of generality, we assume that party P_1 acts as $\mathsf{P}_{\mathsf{leader}}$ in this protocol.

For each $j \in [\frac{N}{B}]$, the parties proceed as follows:

- **Step I: Computing** $(\mathbf{X}_j \times \mathbf{w}_{j-1})$
  1. **Local Computation:** For each $u \in [B/\ell]$, parties locally compute their share of $(\mathbf{z}_j)^{(u)}$ as follows and send the resulting shares to the designated leader $\mathsf{P}_{\mathsf{leader}}$: $\left[\!\left[(\mathbf{z}_j)^{(u)}\right]\!\right]_{2t} = \left[\!\left[(\boldsymbol{\alpha}_j)^{(u)}\right]\!\right]_{2t} + \sum_{k\in[D]} \left[\!\left[\left(\mathbf{X}_j^\intercal\right)^{(u)}_{\langle k,*\rangle}\right]\!\right] \cdot [\![\mathbf{w}_{j-1,k}]\!]$.
  2. **Degree Reduction:** For each $u \in [B/\ell]$, $\mathsf{P}_{\mathsf{leader}}$ reconstructs $(\mathbf{z}_j)^{(u)} \leftarrow \mathsf{Recon}\left(\left[\!\left[(\mathbf{z}_j)^{(u)}\right]\!\right]_{2t}, 2t\right)$. Computes packed shares $\left[\!\left[(\mathbf{z}_j)^{(u)}\right]\!\right] \leftarrow \mathsf{Share}\left(\mathbf{z}_{j,u}, t\right)$ and sends $\left[\!\left[(\mathbf{z}_j)^{(u)}\right]\!\right]^{\mathsf{P}_i}$ to party P_i (for each $i \in [n]$).
  3. **Truncation.** For each $u \in [\frac{B}{\ell}]$, all parties proceeds as follows.
     - Compute and broadcast $[\![(\boldsymbol{a}_j)^{(u)}]\!] = \left[\!\left[(\mathbf{z}_j)^{(u)}\right]\!\right] - \left[\!\left[(\boldsymbol{\alpha}_j)^{(u)}\right]\!\right] + 2^m \cdot \left[\!\left[\mathsf{mask}^{\uparrow}_{j,u}\right]\!\right] + \left[\!\left[\mathsf{mask}^{\downarrow}_{j,u}\right]\!\right]$.
     - Use the broadcasted packed secret shares to reconstruct the secret vector $(\boldsymbol{a}_j)^{(u)} \in \mathbb{F}^\ell$ and locally

compute the shares $[\![(\boldsymbol{b}_j)^{(u)}]\!] = \left([\![(\mathbf{z}_j)^{(u)}]\!] - [\![(\boldsymbol{\alpha}_j)^{(u)}]\!] + [\![\mathsf{mask}^{\downarrow}_{j,u}]\!] - [\![(\boldsymbol{a}_j)^{(u)} \mod 2^m]\!]\right) \cdot 2^{-m}$.

- **Step II: Computing the Activation Function**
  1. **Generate shares of the indicator for the event $(\boldsymbol{b}_j)^{(u)} < -1/2$ and $(\boldsymbol{b}_j)^{(u)} < 1/2$:**
     - For each $u \in [\frac{B}{\ell}]$, parties compute and broadcast $[\![(\boldsymbol{b'}_j)^{(u)}]\!] = [\![(\boldsymbol{b}_j)^{(u)} + 2^M + 1/2]\!] + 2^M \cdot [\![\mathsf{LT}^{\uparrow}_{j,u}]\!] + [\![\mathsf{LT}^{\downarrow}_{j,u}]\!]$ and $[\![(\boldsymbol{b''}_j)^{(u)}]\!] = [\![(\boldsymbol{b}_j)^{(u)} + 2^M - 1/2]\!] + 2^M \cdot [\![\overline{\mathsf{LT}}^{\uparrow}_{j,u}]\!] + [\![\overline{\mathsf{LT}}^{\downarrow}_{j,u}]\!]$.
     - For each $u \in [\frac{B}{\ell}]$, parties verify that the last M bits of the reconstructed values are 1 and compute the secret shares $[\![\mathbb{1}(-\frac{1}{2})^{(u)}_j]\!] = \left([\![(\boldsymbol{b}_j)^{(u)} + 2^M + 1/2]\!] + [\![\mathsf{LT}^{\downarrow}_{j,u}]\!] - [\![(\boldsymbol{b'}_j)^{(u)} \mod 2^M]\!]\right) \cdot 2^{-M}$ and $[\![\mathbb{1}(\frac{1}{2})^{(u)}_j]\!] = \left([\![(\boldsymbol{b}_j)^{(u)} + 2^M - 1/2]\!] + [\![\overline{\mathsf{LT}}^{\downarrow}_{j,u}]\!] - [\![(\boldsymbol{b''}_j)^{(u)} \mod 2^M]\!]\right) \cdot 2^{-M}$ of the indicator.
  2. **Computing Piecewise Function:** For each $u \in [\frac{B}{\ell}]$, parties do the following.
     - Parties compute and send $[\![\boldsymbol{f}^{(u)}_j]\!]_{2t} = [\![\mathbb{1}(-\frac{1}{2})^{(u)}_j]\!] \cdot \left(1 - [\![\mathbb{1}(\frac{1}{2})^{(u)}_j]\!]\right) + [\![\boldsymbol{\theta}^{(u)}_j]\!]_{2t}$ to $\mathsf{P}_{\mathsf{leader}}$, who reconstructs $\boldsymbol{f}^{(u)}_j$ and re-shares it.
     - Parties reconstruct $[\![\mathbb{1}(-\frac{1}{2}, \frac{1}{2})^{(u)}_j]\!] = [\![\boldsymbol{f}^{(u)}_j]\!] - [\![\boldsymbol{\theta}^{(u)}_j]\!]$ and send $\left(1 - [\![\mathbb{1}(-\frac{1}{2})^{(u)}_j]\!]\right) \cdot 0 + [\![\mathbb{1}(-\frac{1}{2}, \frac{1}{2})^{(u)}_j]\!] \cdot [\![(\boldsymbol{b}_j)^{(u)} + \frac{1}{2}]\!] + [\![\mathbb{1}(\frac{1}{2})^{(u)}_j]\!] \cdot 1 + [\![\boldsymbol{\mu}^{(u)}_j]\!]_{2t}$ to $\mathsf{P}_{\mathsf{leader}}$.
     - $\mathsf{P}_{\mathsf{leader}}$ reconstructs these values and re-shares them as $[\![\boldsymbol{e'}^{(u)}_j]\!]$ and all parties locally reconstruct $[\![\boldsymbol{e}^{(u)}_j]\!] = [\![\boldsymbol{e'}^{(u)}_j]\!] - [\![\boldsymbol{\mu}^{(u)}_j]\!]$.
- **Step III: Repacking Shares** All parties proceed as follows:
  - For each $u \in [\frac{B}{\ell}]$, compute and broadcast $[\![(\mathbf{r'}_j)^{(u)}]\!] = [\![(\boldsymbol{e}_j)^{(u)}]\!] - [\![(\mathbf{y}_j)^{(u)}]\!] + [\![(\boldsymbol{\tau}_j)^{(u)}]\!]_{2t}$.
  - Parties then locally reconstruct $(\mathbf{r'}_j)^{(u)}$ and for each $k \in [B]$, they compute deterministic shares $(\mathbf{r'}_j)^{(u)} \leftarrow \mathsf{Recon}\left([\![(\mathbf{r'}_j)^{(u)}]\!]_{2t}, 2t\right)$.
  - For each $k \in [B]$, parties locally compute $[\![\mathbf{r}_{j,k}]\!] = [\![\mathbf{r'}_{j,k}]\!] - [\![\boldsymbol{\nu}_{j,k}]\!]$.
- **Step IV: Remaining Computation**
  1. **Local Computation:** For each $u \in [D/\ell]$, parties locally compute their share of $(\mathbf{s}_j)^{(u)}$ as follows and send the resulting shares to the designated leader $\mathsf{P}_{\mathsf{leader}}$: $[\![(\mathbf{s}_j)^{(u)}]\!]_{2t} = [\![(\boldsymbol{\gamma}_j)^{(u)}]\!]_{2t} + \sum_{k\in[B]} [\![(\mathbf{X}_j)^{(u)}_{\langle k,*\rangle}]\!] \cdot [\![\mathbf{r}_{j,k}]\!]$.
  2. **Degree Reduction:** For each $u \in [D/\ell]$, $\mathsf{P}_{\mathsf{leader}}$ reconstructs $(\mathbf{s}_j)^{(u)} \leftarrow \mathsf{Recon}\left([\![(\mathbf{s}_j)^{(u)}]\!]_{2t}, 2t\right)$. Computes packed shares $[\![(\mathbf{s}_j)^{(u)}]\!] \leftarrow \mathsf{Share}\left(\mathbf{s}_{j,u}, t\right)$ and sends $[\![(\mathbf{s}_j)^{(u)}]\!]^{\mathsf{P}_i}$ to party P_i (for each $i \in [n]$).
  3. **Truncation:** For each $u \in [\frac{D}{\ell}]$, all parties proceeds as follows.
     - Compute and broadcast $[\![(\boldsymbol{c}_j)^{(u)}]\!] = [\![(\mathbf{s}_j)^{(u)}]\!] - [\![(\boldsymbol{\gamma}_j)^{(u)}]\!] + 2^m \cdot [\![\overline{\mathsf{mask}}^{\uparrow}_{j,u}]\!] + [\![\overline{\mathsf{mask}}^{\downarrow}_{j,u}]\!]$.
     - Use the broadcasted packed secret shares to reconstruct the secret vector $(\boldsymbol{c}_j)^{(u)} \in \mathbb{F}^{\ell}$ and locally compute the shares $[\![(\boldsymbol{d}_j)^{(u)}]\!] = \left([\![(\mathbf{s}_j)^{(u)}]\!] - [\![(\boldsymbol{\gamma}_j)^{(u)}]\!] + [\![\overline{\mathsf{mask}}^{\downarrow}_{j,u}]\!] - [\![(\boldsymbol{c}_j)^{(u)} \mod 2^m]\!]\right) \cdot 2^{-m}$.
  4. **Local Computation:** For each $u \in [\frac{D}{\ell}]$, all parties compute $[\![(\mathbf{w}_j)^{(u)}]\!] = [\![(\mathbf{w}_{j-1})^{(u)}]\!] - \frac{a}{B} \cdot [\![(\boldsymbol{d}_j)^{(u)}]\!]$.

- **Step V: Repacking Shares** All parties proceed as follows:
  - For each $u \in [\frac{D}{\ell}]$, compute and broadcast $\left[\!\left[(\mathbf{w'}_j)^{(u)}\right]\!\right] = \left[\!\left[(\mathbf{w}_j)^{(u)}\right]\!\right] + \left[\!\left[(\boldsymbol{\beta}_j)^{(u)}\right]\!\right]_{2t}$.
  - Parties then locally reconstruct $(\mathbf{w'}_j)^{(u)}$ and for each $k \in [D]$, they compute deterministic shares $(\mathbf{w'}_j)^{(u)} \leftarrow \mathsf{Recon}\left(\left[\!\left[(\mathbf{w'}_j)^{(u)}\right]\!\right]_{2t}, 2t\right)$.
  - For each $k \in [D]$, parties locally compute $[\![\mathbf{w}_{j,k}]\!] = [\![\mathbf{w'}_{j,k}]\!] - [\![\boldsymbol{\delta}_{j,k}]\!]$.

Figure 3: Online Phase of Our MPC protocol.

Lemma 4. *Assuming that the pre-processing phase was implemented honestly by a trusted party and that the leader* $\mathsf{P}_{\mathsf{leader}}$ *is honest, the above MPC protocol is* $(t-\ell)$*-private and* $(n-t)$*-robust.*

Proof of Lemma 4. We argue privacy. The view of the adversary consists of the public communication of the protocol and the local views of the corrupted parties. Public communications are just packed secret shares of random values assuming the correlated setup and, hence, simulatable. The local views of $\leqslant t-\ell$ corrupted parties are uniformly random due to the privacy of the packed secret-sharing scheme. Therefore, the entire view of the adversary is simulatable.

We argue robustness. First, observe that, in the semi-honest setting, all possible choices of the randomness tape of the leader will result in the correct output, i.e., there is no bad randomness. Next, consider the malicious setting and let us fix any randomness tape of the leader $\mathsf{P}_{\mathsf{leader}}$. Observe that the output of the honest parties is a *deterministic* function of their views and the communication transcript of the MPC — which only comprises of parties exchanging packed secret shares. Thus, the adversary may only alter the output of the honest parties by changing the transcript from packed shares of an honest value to packed shares of some incorrect values. If there are $\geqslant t$ honest parties, their packed secret shares will be honest and uniquely define the honest value packed. Consequently, if the adversary sends any incorrect secret shares, the packed secret shares will not be well-formed and the honest parties will abort. In conclusion, if the adversary corrupts $\leqslant n-t$ parties, the honest parties' output will always be correct. □

5 Zero-Knowledge Sub-Routines for Specific Relations

In this section, we present a number of sub-routines for checking some specific relations based on techniques from the SNARKs literature. The MPC protocol described in the previous section requires the pre-processing phase to be implemented by a "trusted" entity. Looking ahead, in our MPC-in-the-head based zero-knowledge proof, when the prover runs this MPC in its head, he will act as the "trusted entity" in the MPC protocol. However, for obvious reasons, the prover cannot be simply trusted to have executed this step honestly. Therefore, before it runs the remaining online phase of the MPC in its head and commits to the views of the virtual parties, it must first convince the verifier that the pre-processing phase of the MPC was executed honestly. The sub-routines presented in this section will be used exactly for this purpose. For reasons that will become clear in Section 6, these proofs are given with respect to shares and values committed in a very specific manner using polynomial commitments. Several of the proofs use the sum-check protocol (see Section 2.6 for reference)as a subroutine.

Notations. For efficiency reasons, SNARK proofs typically work with a specific multiplicative subgroup $\mathbb{H} = \{\omega, \omega^2, \ldots, \omega^{|\mathbb{H}|}\} \subset \mathbb{F}^*$ whose order is a power of 2. This approach offers several advantages, such as FFT-friendly evaluation points and a succinct representation of the Lagrange basis polynomial. We

Sub-Protocol for Checking Consistency of Packed Secret Shares

1. **Round 1. (Prover)** For each $v \in [\ell]$, the prover computes and sends commitments to polynomial q_v^g such that
$$g_v(X) - \sum_{i\in[n]} \mathcal{L}_{i,v}^{\mathbb{Q}}(X) \cdot g^{(i)}(X) = q_v^g(X) \cdot z_{\mathbb{Q}}(X).$$
2. **Round 2. (Verifier)** Verifier sends random field elements $\epsilon, \alpha \in \mathbb{F}$ and a random n-sized vector $\boldsymbol{r} = (r_1, \ldots, r_n)$ from the dual space of Reed-Solomon codes (see Section 2.1 for reference) to the prover.
3. **Round 3. (Prover)** Prover sends openings to the following polynomial evaluations: $\forall v \in [\ell]$: $g_v(\epsilon), q_v^g(\epsilon)$ and $\forall i \in [n], g^{(i)}(\epsilon)$.
4. **Round 3-5 (Sum-Check)** Computes $g(X) = \sum_{i\in[n]} r_i \cdot g^{(i)}(X)$. Let $b(X)$ be a polynomial encoding of a vector $\boldsymbol{\beta} = (\alpha^0, \ldots, \alpha^{\mathsf{s}-1})$. Prover invokes a modified instance of the sumcheck protocol (see Section 2.6 for a detailed discussion on how to prove such a relation) to prove that: *sum of evaluations of $g(X) \cdot b(X)$ on s^{th} roots of unity is 0.*
5. **Local Verification** Verifier checks if all the openings are correct and checks if the following holds for each $v \in [\ell]$:
$$g_v(\epsilon) - \sum_{i\in[n]} \mathcal{L}_{i,v}^{\mathbb{Q}}(\epsilon) \cdot g^{(i)}(\epsilon) = q_v^g(\epsilon) \cdot z_{\mathbb{Q}}(\epsilon).$$

Output. If all the checks in the above protocol verify, the verifier outputs accept, else it outputs reject.

Figure 4: Checking for Consistency of Packed Secret Shares

usually use ω as the generator of the subgroup. However, in some cases, we may need to work with multiple subgroups of different orders. In such situations, we use μ, φ, ψ as the generators of those other subgroups. The orders of the subgroups generated by the corresponding generators shall be clear from the context.

5.1 Checking Consistency of Packed Secret Shares

At the end of the pre-processing phase, all parties receive packed secret shares of some random vectors and the data. The following sub-protocol allows the prover to succinctly prove to the verifier that the shares sent to the parties correspond to valid packed secret sharings of some vectors. For technical reasons, we give both the commitment to the secret shares and the commitment to the actual secrets. We use n different commitments for the shares, one corresponding to each party, and ℓ commitments to store the actual secrets. Through this sub-protocol, we check the following: (1) the commitment to the secret shares are well-formed secret shares and (2) the reconstructions of the secret shares are consistent with the secrets committed.

Lemma 5. *If* $\deg > \mathsf{s} - 1$*, then the protocol in Figure 4 is a correct, sound, and zero-knowledge proof system for the following relation* $\mathcal{R}_{shares}$*:*

- ***Statement:*** *Commitments to the following degree* $\deg$ *polynomials:* $\forall i \in [n], g^{(i)}(X)$ *and* $\forall v \in [\ell], g_v(X)$.
- ***Witness:*** $\boldsymbol{g} \in \mathbb{F}^{\ell \cdot \mathsf{s}}$ *and* $\forall i \in [n]$*, polynomials* $g^{(i)}(X)$.
- ***Relation:*** $\forall a \in [0, \mathsf{s} - 1]$: $g_v(\omega^a) = \boldsymbol{g}[a\ell + v]$ *and* $(g^{(1)}(\omega^a), \ldots, g^{(n)}(\omega^a))$ *are packed shares of* $(\boldsymbol{g})^{(a)}$.

Sub-Protocol for Checking Consistency of Correlated Randomness

1. **Round 1. (Verifier)** Verifier sends random field elements $\{\rho_a\}_{a\in[\mathsf{s}-1]} \in \mathbb{F}$ to the prover.
2. Both the prover and verifier compute the following:
   - A polynomial $k(X)$ of degree $\mathsf{s}-1$ such that for each $a \in [\mathsf{s}-1]$, $k(\omega^{a-1}) = \rho_a$. Let $h'(X) = k(X) \cdot h(X)$.
   - For each $v \in [\ell]$, a polynomial $f_v(X)$ of degree $\frac{\mathsf{s}}{\ell} - 1$ such that for each $a \in [0, \frac{\mathsf{s}}{\ell} - 1]$, $f_v(\mu^a) = \rho_{\ell\cdot a+v}$. Let $h'_v(X) = f_v(X) \cdot h_v(X)$.
3. **Round 2-4 (Sum-Check)** Let $h'(X) = \sum_{v\in[\ell]} h'_v(X)$ and $g'(X) = \sum_{v\in[\ell]} g'_v(X)$. Prover invokes a modified instance of the sumcheck protocol (see Section 2.6 for a detailed discussion on how to prove such a relation) to prove that: *sum of evaluations of $h'(X)$ on s^{th} roots of unity - sum of evaluations of $g'(X)$ on s/ℓ^{th} roots of unity is 0.*

Output. If all the checks in the above sum-check protocols verify, the verifier outputs accept, else it outputs reject.

Figure 5: Checking Consistency of Correlated Randomness

Proof of Lemma 5. Correctness is straightforward. The zero-knowledge property comes from the fact that the view of the verifier consists only of a number of openings of the polynomials. Since all the polynomials are randomized encodings, these openings are all random field elements, which are simulatable.

We argue soundness. Let

$$G = \begin{pmatrix} g^{(1)}(\omega^1) & g^{(1)}(\omega^2) & \cdots & g^{(1)}(\omega^{\mathsf{s}}) \\ \vdots & \vdots & \ddots & \vdots \\ g^{(n)}(\omega^1) & g^{(n)}(\omega^2) & \cdots & g^{(n)}(\omega^{\mathsf{s}}) \end{pmatrix}.$$

As we discussed in Section 2.1, the secret shares are well-formed as long as it passes the parity check. For a random vector r spanned by the parity check matrix, if $g^{(i)}$ are not well-formed secret shares, $r \cdot G$ will not be a zero-vector with probability $1 - \frac{\mathsf{s}}{|\mathbb{F}|}$. Conditioned on it being a non-zero vector, $r \cdot G \cdot \left(\vec{\beta}\right)^\mathsf{T}$ is not 0 with probability $1 - \frac{\mathsf{s}}{|\mathbb{F}|}$ by Schwartz-Zippel. Finally, if $r \cdot G \cdot \left(\vec{\beta}\right)^\mathsf{T} \neq 0$, the verifier will reject the proof with overwhelming probability by the soundness of the sum-check protocol.

The soundness for the consistency between the secret shares and the secrets are straightforward. The polynomials $g^{(1)}, \ldots, g^{(n)}$ and g_v should satisfy the linear relationship defined by the Lagrange interpolation. By Schwartz-Zippel, if this linear relation is satisfied at a random point $x = \epsilon$, it holds globally with overwhelming probability. Finally, the soundness follows from the soundness of the polynomial commitment scheme, which guarantees that the openings of the polynomial are correct with overwhelming probability. □

5.2 Checking Consistency of Correlated Randomness

The following sub-routine allows the prover to convince the verifier that $\boldsymbol{\tau}$ and $\boldsymbol{\nu}$ values (and $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$ values, resp.) are appropriately correlated (i.e., these are the same values that were secret shared using different thresholds). Looking ahead, this sub-routine will be invoked together with the sub-routine from Section 5.1 to ensure that the shares of these random values are consistent and appropriately correlated. As before, for technical reasons, we store these values across ℓ polynomial commitments.

Lemma 6. *If* $\mathsf{deg}_1 > \mathsf{s}/\ell - 1$ *and* $\mathsf{deg}_2 > \mathsf{s} - 1$*, then the protocol in Figure 5 is a correct, sound, and zero-knowledge proof system for the following relation* $\mathcal{R}_{corr}$*:*

- ***Statement:*** $\forall v \in [\ell]$*, commitments to degree* deg_1 *polynomial* $g_v(X)$ *and a degree* deg_2 *polynomial* $h(X)$*.*
- ***Witness:*** $\boldsymbol{g} \in \mathbb{F}^{\mathsf{s}}$.
- ***Relation****: The following conditions hold:*
  - $\forall a \in [\mathsf{s}], h(\omega^a) = \boldsymbol{g}[a]$
  - $\forall v \in [\ell], \forall a \in [0, \frac{\mathsf{s}}{\ell} - 1], g_v(\mu^a) = \boldsymbol{g}[a\ell + v]$.

Proof of Lemma 6. Correctness is straightforward. The zero-knowledge property follows from the fact that the verifier's view consists of a number of openings of polynomials (in the sum-check protocol). Since the polynomials are randomized encoded, all the openings are random field elements and, hence, are simulatable.

We argue soundness. Intuitively, to prove that two vectors $\vec{a}$ and $\vec{b}$ are identical. The verifier picks a random $\vec{\rho}$, and the prover proves that $\vec{\rho} \cdot \vec{a} - \vec{\rho} \cdot \vec{b} = 0$. This is indeed what this protocol does. The verifier picks the vector $\vec{\rho} = (\rho_0, \ldots, \rho_{s-1})$ and generates a polynomial $k(X)$ that encodes $\vec{\rho}$. The product $h'(X)$ between $k(X)$ and $h(X)$ encodes the coordinate-wise products between the vectors. Similarly, $g'_v(X)$ encodes the coordinate-wise products between $\vec{\rho}$ and the other vector. By the soundness of the sum-check protocol, we know that with an overwhelming probability, the difference between the sums of the values encoded in those polynomials is 0. By our earlier argument, with $1 - \frac{1}{|\mathbb{F}|}$ probability, the statement holds. □

5.3 Checking the Range for Truncation Masks

To ensure correctness, all masks used in the main MPC protocol for non-arithmetic operations must come from a certain range. The subprotocol in this section allows the prover to generate a succinct proof for this. The protocol utilizes similar ideas as in the bit-decomposition-based range proofs (e.g., [CCs08, BBB+18]). Intuitively, to prove that $0 \leqslant x < 2^{\mathsf{range}}$, one commits to the binary representation $x_0, x_1, \ldots, x_{\mathsf{range}-1}$ and proves two conditions: (1) for all i, $x_i^2 - x_i = 0$ (i.e., $x_i \in \{0, 1\}$) and (2) $x = \sum_i 2^i \cdot x_i$.

Lemma 7. *If* $\mathsf{deg} > \frac{\mathsf{s}}{\ell} - 1$*, then the protocol in Figure 6 is a correct, sound, and zero-knowledge proof system for the following relation* $\mathcal{R}_{mask}$*:*

- ***Statement:*** $\forall v \in [\ell]$*, commitments to degree* deg *polynomials* $p_v(X)$ *and a range* range.
- ***Witness:*** $\boldsymbol{p} \in \mathbb{F}^{\mathsf{s}}$.
- ***Relation****: The following conditions hold:*
  - $\forall v \in [\mathsf{s}], \boldsymbol{p}[v] \in [0, 2^{\mathsf{range}})$.
  - $\forall v \in [\ell], \forall a \in [0, \frac{\mathsf{s}}{\ell} - 1], p_v(\mu^a) = \boldsymbol{p}[a\ell + v]$.

Proof of Lemma 7. Correctness is straightforward. The zero-knowledge property comes from the fact that the view of the verifier consists only of a number of openings of the polynomials $\mathsf{bin}_v^c(X)$, $p_v(X)$, and $q_{c,v}(X)$. Since all the polynomials are randomized encodings, these openings are all random field elements, which are simulatable.

Sub-Protocol for Checking the Range for Truncation Masks

1. **Round 1. (Prover)** For each $u \in [\mathsf{s}]$, let $b_{u,1}, \ldots, b_{u,\mathsf{range}}$ denote the bit decomposition of $\boldsymbol{p}[u]$. The prover computes and sends commitments to the following polynomials for each $v \in [\ell]$:
   - For each $c \in [\mathsf{range}]$, a randomized polynomial bin_v^c of degree $\frac{\mathsf{s}}{\ell}$, such that for each $a \in [0, \frac{\mathsf{s}}{\ell} - 1]$, $\mathsf{bin}_v^c(\mu^a) = b_{a\ell+v,c}$.
   - For each $c \in [\mathsf{range}]$, polynomial $q_{c,v}$ such that $\mathsf{bin}_v^c(X) \cdot \mathsf{bin}_v^c(X) - \mathsf{bin}_v^c(X) = q_{c,v}(X) \cdot z_{\mathbb{S}}(X)$.
2. **Round 2. (Verifier)** Verifier sends random field elements $\epsilon \in \mathbb{F}$ to the prover.
3. **Round 3. (Prover)** Prover sends openings to the following polynomial evaluations:
   - For each $v \in [\ell]$: $p_v(\epsilon)$.
   - For each $v \in [\ell]$: for each $c \in [\mathsf{range}]$, $\mathsf{bin}_v^c(\epsilon)$, $q_{c,v}(\epsilon)$.
4. **Local Verification** The verifier checks if all the openings are correct and performs the following checks for each $v \in [\ell]$:
   - $p_v(\epsilon) = \sum_{j \in [\mathsf{range}]} 2^{j-1} \cdot \mathsf{bin}_v^j(\epsilon)$.
   - For each $c \in [\mathsf{range}]$, $\mathsf{bin}_v^c(\epsilon) \cdot \mathsf{bin}_v^c(\epsilon) - \mathsf{bin}_v^c(\epsilon) = q_{c,v}(\epsilon) \cdot z_{\mathbb{S}}(\epsilon)$.

Output. If all of these checks verify, the verfier outputs accept, else it outputs reject.

Figure 6: Checking the Range for Truncation Masks

We argue soundness. By the soundness of the polynomial commitment scheme, with overwhelming probability, bin_v^c, p_v, and $q_{c,v}$ are polynomials with the correct opening $\mathsf{bin}_v^c(\epsilon)$, $p_v(\epsilon)$, and $q_{c,v}(\epsilon)$. Next, by Schwartz-Zippel lemma, with probability $\geqslant 1 - \frac{2\deg}{|\mathbb{F}|}$, the following polynomial identities hold for all $v \in [\ell]$.

$$p_v(X) = \sum_{j \in [\mathsf{range}]} 2^{j-1} \cdot \mathsf{bin}_v^j(X)$$
$$\mathsf{bin}_v^c(X) \cdot \mathsf{bin}_v^c(X) - \mathsf{bin}_v^c(X) = q_{c,v}(X) \cdot z_{\mathbb{S}}(X)$$

The second polynomial identity guarantees that $\mathsf{bin}_v^c(X) \in \{0, 1\}$ for all $X \in \mathbb{S}$. This fact and the first polynomial identity guarantee that, for all $X \in \mathbb{S}$, $p_v(X) \in [0, 2^{\mathsf{range}})$. □

5.4 Checking Consistency of Data Shares

In our MPC protocol, we require parties to have two kinds of packed secret sharings of the data. Therefore, it must be proved that these two sharings are consistent with each other. For technical reasons, we give the polynomial commitments to both $X_1, \ldots, X_{N/B}$ and $X_1^\intercal, \ldots, X_{N/B}^\intercal$. The sub-protocol in the section essentially allows the prover to give a succinct proof to convince the verifier that these commitments satisfy the "transpose" relationship.

Intuitively, to prove that two matrices A and B satisfy $A = B^\intercal$, we simply pick random vectors (of appropriate dimensions) $\vec{\alpha}$ and $\vec{\beta}$, and checks

$$\vec{\alpha} \cdot A \cdot \left(\vec{\beta}\right)^\intercal \stackrel{?}{=} \vec{\beta} \cdot B \cdot (\vec{\alpha})^\intercal .$$

Sub-Protocol for Checking Consistency of Data Shares

1. **Round 1. (Verifier)** The verifier sends random field elements $\alpha_1, \ldots, \alpha_D, \beta_1, \ldots, \beta_B, r \in \mathbb{F}$ to the prover.
2. Prover computes the following:
   - $p^{\text{row}}(X) = \sum_{u \in [D/\ell], v \in [\ell]} \alpha_{\ell(u-1)+v} \cdot \text{row}_{u,v}(X)$.
   - A polynomial f of degree $N-1$, such that for each $a \in [B], b \in [0, N/B - 1]$, $f(\varphi^{\frac{N}{B}(a-1)+b}) = \beta_a \cdot r^{b+1}$. Let $s^{\text{row}}(X) = f(X) \cdot p^{\text{row}}(X)$.
   - $p^{\text{col}}(X) = \sum_{u \in [B/\ell], v \in [\ell]} \beta_{\ell(u-1)+v} \cdot \text{col}_{u,v}(X)$.
   - A polynomial k of degree $ND/B - 1$, such that for each $a \in [D], b \in [0, N/B - 1]$, $k(\psi^{N/B(a-1)+b}) = \alpha_a \cdot r^{b+1}$. Let $s^{\text{col}}(X) = k(X) \cdot p^{\text{col}}(X)$.
3. **Round 2-4 (Sum-Check)** Prover invokes a modified instance of the sumcheck protocol (see Section 2.6 for a detailed discussion on how to prove such a relation) to prove that: *sum of evaluations of $s^{\text{row}}(X)$ on N^{th} roots of unity - sum of evaluations of $s^{\text{col}}(X)$ on ND/B^{th} roots of unity is 0.*

Output. If all the checks in the above sum-check protocols verify, the verifier outputs accept, else it outputs reject.

Figure 7: Checking Consistency of Data Shares within a given iteration

If this check passes, with probability $\geqslant 1 - \frac{1}{|\mathbb{F}|}$, it holds that $A = B^\intercal$.

For our purpose, we have two matrices

$$A = \left(X_1, \ldots, X_{N/B}\right)^\intercal \qquad\qquad B = \left(X_1^\intercal, \ldots, X_{N/B}^\intercal\right)^\intercal.$$

To checks that A and B indeed satisfy this "special-transpose" relationship, we picks $\vec{\alpha}$, $\vec{\beta}$, and additionally a random $r \in \mathbb{F}$. We check if

$$\left(r \cdot \vec{\alpha}, r^2 \cdot \vec{\alpha}, \ldots, r^{N/B} \cdot \vec{\alpha}\right) A \vec{\beta} \overset{?}{=} \left(r \cdot \vec{\beta}, r^2 \cdot \vec{\beta}, \ldots, r^{N/B} \cdot \vec{\beta}\right) B \vec{\alpha}.$$

If this passes, by Schwartz-Zippel lemma, with $1 - \frac{\text{poly}(\lambda)}{|\mathbb{F}|}$ probability, the relationship is satisfied.

Lemma 8. *If* $\deg_1 > N - 1$ *and* $\deg_2 > ND/B - 1$*, then the protocol in Figure 7 is a correct, sound, and zero-knowledge proof system for the following relation* $\mathcal{R}_{data}$*:*

- ***Statement:*** $\forall v \in [\ell]$, $u \in [D/\ell]$, *commitment to degree* $\deg_1$ *polynomial* $\text{row}_{u,v}(X)$ *and* $\forall v \in [\ell]$, $u \in [B/\ell]$, *commitment to degree* $\deg_2$ *polynomial* $\text{col}_{u,v}(X)$
- ***Witness:*** $\forall j \in [N/B]$, $\mathbf{X}_j \in \mathbb{F}^{B \times D}$.
- ***Relation****: The following conditions hold* $\forall v \in [\ell]$*:*
  - $\forall u \in [D/\ell]$, $\text{row}_{u,v}(X)$ *is a polynomial encoding of the concatenation the vectors* $(\mathbf{X}_1)_{\langle *, (u-1)\cdot\ell+v \rangle}, \ldots, (\mathbf{X}_{N/B})_{\langle *, (u-1)\cdot\ell+v \rangle}$.
  - $\forall u \in [B/\ell]$, $\text{col}_{u,v}(X)$ *is a polynomial encoding of the concatenations of the vectors* $(\mathbf{X}_1^\intercal)_{\langle *, (u-1)\cdot\ell+v \rangle}, \ldots, (\mathbf{X}_{N/B}^\intercal)_{\langle *, (u-1)\cdot\ell+v \rangle}$.

Proof of Lemma 8. Correctness is straightforward. The zero-knowledge property comes from the fact that the view of the verifier, during the sum-check protocol, consists only of a number of openings of the polynomials. Since all the polynomials are randomized encodings, these openings are all random field elements, which are simulatable.

We argue soundness. By the soundness of the sum-check protocol, with overwhelming probability, we have that

$$\left(r \cdot \vec{\alpha}, r^2 \cdot \vec{\alpha}, \ldots, r^{N/B} \cdot \vec{\alpha}\right) A \vec{\beta} \overset{?}{=} \left(r \cdot \vec{\beta}, r^2 \cdot \vec{\beta}, \ldots, r^{N/B} \cdot \vec{\beta}\right) B \vec{\alpha}, \tag{2}$$

where $\vec{\alpha} = (\alpha_1, \ldots, \alpha_D)$, $\vec{\beta} = (\beta_1, \ldots, \beta_B)$, and

$$A = \left(\mathsf{row}_{1,1} \| \cdots \| \mathsf{row}_{1,\ell} \| \cdots \| \mathsf{row}_{D/\ell,1} \| \cdots \| \mathsf{row}_{D/\ell,\ell}\right),$$
$$B = \left(\mathsf{col}_{1,1} \| \cdots \| \mathsf{col}_{1,\ell} \| \cdots \| \mathsf{col}_{B/\ell,1} \| \cdots \| \mathsf{col}_{B/\ell,\ell}\right).$$

It remains to show that Identity 2 implies the "special-transpose" relation between $\{\mathsf{row}_{u,v}\}$ and $\{\mathsf{col}_{u,v}\}$. Let us write

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_{N/B} \end{pmatrix} \qquad B = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_{N/B} \end{pmatrix}.$$

Furthermore, let us write

$$c_i = \vec{\alpha} \cdot (A_i - B_i^\top) \cdot (\vec{\beta})^\top.$$

If there exists an i such that $A_i \neq B_i^\top$, then $c_i \neq 0$ with probability $1 - \frac{1}{|\mathbb{F}|}$. Then $\sum_i c_i \cdot r^i \neq 0$ with probability $1 - \frac{N/B}{|\mathbb{F}|}$ by Schwartz-Zippel. Therefore, Identity 2 implies, with overwhelming probability, that $\{\mathsf{row}_{u,v}\}$ and $\{\mathsf{col}_{u,v}\}$ satisfies the "special-transpose" relation. □

6 Zero-Knowledge Proof of Training for Logistic Regression

With all the tools presented in the previous two sections, we are now ready to describe our zero-knowledge proof of training for logistic regression. We start by giving an overview and then proceed to give a formal description.

6.1 Overview

Based on the discussion thus far, it is clear that we want to use the MPC protocol from Section 4 to instantiate an MPC-in-the-head framework, where we somehow want to use the zkSNARK sub-protocols from Section 5 to verify whether the prover honestly executed the pre-processing phase in the MPC. To verify whether the online phase of this MPC was honestly executed, we still want to primarily rely on MPC-in-the-head style techniques. However, as discussed towards the end of Section 4.1, our MPC protocol cannot be used as is to obtain a sublinear-sized zero-knowledge proof.

We now discuss our final idea to reduce the communication complexity of our MPC-in-the-head-based zero-knowledge proof of training. Recall that the reason why the verifier needs to learn the entire view of the opened virtual parties in an MPC-in-the-head protocol, is to perform some local "consistency-checks". At a high level, these checks ensure that each of these opened parties performed the computation honestly based on the messages that they receive from other virtual parties and that the messages exchanged between the opened parties are consistent with each other. In other words, these consistency checks can be divided into — (1) ones that check for the correctness of computation and (2) ones that check communication consistency.

Since the communication complexity of our underlying MPC is $O(N)$, the complexity of checks that ensure communication consistency is also $O(N)$. While the complexity of checks that ensure consistency of computation is $O(DN)$. Our aim is to reduce the complexity of this class of checks to $O(N)$ in the hope that instead of sending the views of the opened virtual parties in the clear to the verifier, it somehow suffices for us to give a functional commitment to a major chunk of the opened parties' views to the verifier. And the verifier is still able to perform the checks using these functional commitments with $O(N)$ instead of $O(DN)$ complexity.

Towards this, we make the following observations:

- The most significant chunk of each party's view is a sharing of the initial input data, i.e., $\overrightarrow{X}_1, \ldots, \overrightarrow{X}_N \in \mathbb{R}^D$. And this is only used by the verifier to check for computation correctness (i.e., consistency checks of the first type). The rest of the view is only of size $O(N)$. Therefore, we can simply focus on giving some kind of functional commitment to these shares of the data and modifying the corresponding consistency checks to reduce their complexity to $O(N)$ instead of $O(DN)$. The hope is to use interaction (with the prover) during the verifier's "consistency-checking phase".

- Our second observation is that the checks that the verifier performs w.r.t. to the shares of $\overrightarrow{X}_1, \ldots, \overrightarrow{X}_N \in \mathbb{R}^D$ are essentially of the following form: Given some $\overrightarrow{a}_i, \overrightarrow{b}_i, c$, check if $\langle \overrightarrow{a}_i, \overrightarrow{b}_i \rangle \overset{?}{=} c$, where $\overrightarrow{a}_i, \overrightarrow{b}_i$ can be derived via some linear function on the shares of $\overrightarrow{X}_1, \ldots, \overrightarrow{X}_N \in \mathbb{R}^D$.

Our Idea: Our idea here is to combine all of these checks for all into a single check. In particular, embed all the $\overrightarrow{a}_i$s into a vector of polynomials (say $\overrightarrow{A}(x)$) using the polynomial extension. Similarly, embed all the $\overrightarrow{b}_i$s into another vector of polynomials (say $\overrightarrow{B}(x)$) and all the c_is into a polynomial $C(x)$. To ensure that the inner product relation is satisfied for all $(\overrightarrow{a}_i, \overrightarrow{b}_i, c)$ triples, it now suffices (due to Schwartz-Zippel Lemma) to check if the inner-product relation is satisfied at the random point on polynomials $\overrightarrow{A}(x), \overrightarrow{B}(x), C(x)$. For this, it suffices to the prover to give a polynomial commitment to polynomials $\overrightarrow{A}(x), \overrightarrow{B}(x), C(x)$ for the verifier to query them at a random point and check if the inner product relation is satisfied.

Overall, this helps us reduce the size of our proof to $O(N)$, even though the total computation in logistic regression is $O(DN)$.

6.2 Protocol Description

We now give a formal description of our zero-knowledge proof of training for logistic regression. As discussed in Section 3, we assume that the verifier has commitments to the final model (aka the final weights vector), the randomness used in the training process as well as the dataset. Throughout this protocol, the prover computes commitments to polynomial encodings (see Section 2.6) of shares of held by the parties in the MPC and of some intermediate values in the computation (that might be witness dependent.) In order to guarantee the zero-knowledge property and ensure that these commitments do not leak the witness or model when the verifier queries them at random points, the prover sends commitments to *randomized* encodings. This is a standard trick used in zkSNARKs. The amount of randomness added to each encoding depends on the number of times the verifier queries the polynomial commitment.

Our Main Protocol: For each iteration, the prover and verifier proceed as follows:

- **Data Shares:** Let the input training sample be $\mathbf{X}_j \in \mathbb{F}^{B \times D}, \forall j \in [N/B]$ and the corresponding output be $y_1, \ldots, y_N \in \mathbb{F}$.

1. **Packed Shares:** $\forall j \in [N/B]$, prover computes packed secret shares: $\left\{ \left[\!\left[(\mathbf{y}_j)^{(u)} \right]\!\right], \left\{ \left[\!\left[\left(\mathbf{X}_j^{\mathsf{T}}\right)^{(u)}_{\langle k,*\rangle} \right]\!\right] \right\}_{k\in[D]} \right\}_{u\in[B/\ell]}$ and $\left\{ \left[\!\left[(\mathbf{X}_j)^{(u)}_{\langle k,*\rangle} \right]\!\right] \right\}_{k\in[B],u\in[D/\ell]}$ in the pre-processing phase in the MPC protocol.
2. **Polynomial Commitments:**
   - Encoding Party Shares: For each party $i \in [n]$: for each $\forall u \in [D/\ell]$, the prover sends a commitment to a *randomized* polynomial encoding $\mathsf{row}^{(i)}_u(X)$ of *all* shares of the form $\left[\!\left[(\mathbf{X}_j)^{(u)}_{\langle k,*\rangle} \right]\!\right]^{\mathsf{P}_i}$. Similarly for each $u \in [B/\ell]$, compute and send commitments to randomized encodings $\mathsf{col}^{(i)}_u(X)$ for shares corresponding to $\mathbf{X}_j^{\mathsf{T}}$.
   - Encoding Data: For each $v \in [\ell]$: for each $\forall u \in [D/\ell]$, we assume that the prover and verifier have a commitment to a polynomial encoding $\mathsf{row}_{u,v}(X)$ of the v^{th} element in $\left(\mathbf{X}_j^{\mathsf{T}}\right)^{(u)}_{\langle k,*\rangle}$ for each $k \in [B], j \in [N/B]$. Similarly for each $u \in [B/\ell]$, compute encodings $\mathsf{col}_{u,v}(X)$ for corresponding elements in $\mathbf{X}_j^{\mathsf{T}}$. *This corresponds to the commitment to the data that the verifier has at the beginning of the protocol.*
3. **Checking Well-Formedness of Packed Shares:** Invoke instances of the sub-protocol from Figure 4 on the following statements: $\forall u \in [D/\ell]$, $\left(\left\{\mathsf{row}_{u,v}\right\}_{v\in[\ell]}, \left\{\mathsf{row}^{(i)}_u\right\}_{i\in[n]} \right)$ and $\forall u \in [B/\ell]$, $\left(\left\{\mathsf{col}_{u,v}\right\}_{v\in[\ell]}, \left\{\mathsf{col}^{(i)}_u\right\}_{i\in[n]} \right)$.
4. **Checking Consistency of Row and Column Encodings:** Invoke an instance of the sub-protocol from Figure 7 on statement $\left(\left\{ \{\mathsf{row}_{u,v}\}_{u\in[D/\ell]}, \{\mathsf{col}_{u,v}\}_{u\in[B/\ell]} \right\}_{v\in[\ell]} \right)$.
5. **Checking Consistency of Data with Previous Iteration:** Let the encodings of data in the previous iteration were of the form $\hat{\mathsf{row}}_{u,v}$ and $\hat{\mathsf{col}}_{u,v}$. Invoke an instance of the permutation check on statement $\left\{ \{\mathsf{row}_{u,v}, \hat{\mathsf{row}}_{u,v}\}_{u\in[D/\ell]}, \{\mathsf{col}_{u,v}, \hat{\mathsf{col}}_{u,v}\}_{u\in[B/\ell]} \right\}_{v\in[\ell]}$ and the permutation polynomial used during the training process. *The commitments to this permutation polynomial essentially corresponds to the commitment to randomness that the verifier receives at the beginning of the protocol.*

- **Pre-Processed Random Sharings:** Following is done to generate and ensure consistency of random shares generated in the pre-processing phase in the MPC:

1. **Packed Shares:** Prover samples random vectors of appropriate lengths in the pre-processing phase of the MPC (as described in Section 4.2) and generates the following packed secret shares $\forall j \in [N/B]$: $\left\{ \left[\!\left[(\boldsymbol{\alpha}_j)^{(u)} \right]\!\right] \right\}_{u\in[B/\ell]}, \left\{ \left[\!\left[\left(\boldsymbol{\gamma}_j\right)^{(u)} \right]\!\right] \right\}_{u\in[D]/\ell}, \{ [\![\boldsymbol{\nu}_{j,k}]\!] \}_{k\in[B]}, \left\{ \left[\!\left[(\boldsymbol{\alpha}_j)^{(u)} \right]\!\right]_{2t} \right\}_{u\in[B/\ell]}, \left\{ \left[\!\left[\left(\boldsymbol{\gamma}_j\right)^{(u)} \right]\!\right]_{2t} \right\}_{u\in[D/\ell]}$, $\left\{ \left[\!\left[(\boldsymbol{\tau}_j)^{(u)} \right]\!\right]_{2t} \right\}_{u\in[B/\ell]}, \left\{ \left[\!\left[(\boldsymbol{\theta}_j)^{(u)} \right]\!\right]_{2t}, \left[\!\left[\left(\boldsymbol{\mu}_j\right)^{(u)} \right]\!\right]_{2t}, \left[\!\left[(\boldsymbol{\theta}_j)^{(u)} \right]\!\right], \left[\!\left[\left(\boldsymbol{\mu}_j\right)^{(u)} \right]\!\right] \right\}_{u\in[B/\ell]}$.
2. **Polynomial Commitments:** Prover sends commitments to the following *randomized* polynomial encodings:
   - $\forall i \in [n]$, a randomized encoding $p^{(i)}(X)$ of vector $\left(\left[\!\left[(\boldsymbol{\alpha}_j)^{(u)} \right]\!\right]^{\mathsf{P}_i}_{2t} \right)_{j\in[N/B],u\in[B/\ell]}$.
   - $\forall v \in [\ell]$, a randomized encoding $p_v(X)$ of vector $\left((\boldsymbol{\alpha}_j)^{(u)}\,[v] \right)_{j\in[N/B],u\in[B/\ell]}$.

Compute and send commitments to similar randomized encodings for both types of sharings of $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}, \boldsymbol{\theta}$, $\boldsymbol{\mu}, \boldsymbol{\tau}$, $\boldsymbol{\nu}$ and their actual values. Compute and send commitments to randomized polynomial encodings of vectors $(\boldsymbol{\tau}_j)_{j\in[N/B]}$ (and $(\boldsymbol{\beta}_j)_{j\in[N/B]}$ resp.) and for each $i \in [n]$, compute an encoding of vector $\left(\llbracket \boldsymbol{\nu}_{j,k} \rrbracket^{\mathsf{P}_i}\right)_{j\in[N/B],k\in[B]}$ (and $\left(\llbracket \boldsymbol{\delta}_{j,k} \rrbracket^{\mathsf{P}_i}\right)_{j\in[N/B],k\in[D]}$ resp.).

3. **Checking Well-Formedness of Packed Shares:** Similar to how it was done for data shares, the prover and verifier invoke instances of the sub-protocol from Figure 4 on the above encodings of shares and values to check for well-formedness of all packed secret sharings of each type of randomness in one-shot (For instance, the well-formedness of all shares of all α-values is checked in one-shot). We note that when checking the well-formedness of both types of shares of $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$, $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ values, we use the same encodings of the actual $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$, $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ values. This also helps ensure that the two types of sharings are also consistent with each other. For checking well-formedness of the shares of $\boldsymbol{\nu}_{j,k}$, we use ℓ copies of the same polynomial encoding of vector $(\boldsymbol{\tau}_j)_{j\in[N/B]}$.
4. **Checking Consistency of Correlated Randomness** Invoke 2 instances of the sub-protocol from Figure 5 on encodings of $\boldsymbol{\tau}$, $\boldsymbol{\nu}$ and $\boldsymbol{\beta}$, $\boldsymbol{\delta}$ respectively.

- **Masks for Non-Arithmetic Computations:** Following is done to generate shares of masks (and ensure that they come from the right range) that will be used for non-arithmetic operations in the online phase of the MPC:

1. **Packed Shares:** Prover samples random masks of appropriate lengths and from the appropriate range in the pre-processing phase of the MPC (as described in Section 4.2) and generates the following packed secret shares for each $j \in [N/B]$: $\left\{\left\llbracket \mathsf{mask}^{\downarrow}_{j,u} \right\rrbracket, \left\llbracket \mathsf{mask}^{\uparrow}_{j,u} \right\rrbracket\right\}_{u\in[B/\ell]}$, $\left\{\left\llbracket \mathsf{LT}^{\downarrow}_{j,u} \right\rrbracket, \left\llbracket \mathsf{LT}^{\uparrow}_{j,u} \right\rrbracket\right\}_{u\in[B/\ell]}$, $\left\{\left\llbracket \overline{\mathsf{LT}}^{\downarrow}_{j,u} \right\rrbracket, \left\llbracket \overline{\mathsf{LT}}^{\uparrow}_{j,u} \right\rrbracket\right\}_{u\in[B/\ell]}$ and $\left\{\left\llbracket \overline{\mathsf{mask}}^{\downarrow}_{j,u} \right\rrbracket . \left\llbracket \overline{\mathsf{mask}}^{\uparrow}_{j,u} \right\rrbracket\right\}_{u\in[D/\ell]}$.
2. **Polynomial Commitments:** As before, prover computes and sends commitments to randomized polynomial encodings of the shares/values of masks $\mathsf{mask}^{\downarrow}, \mathsf{mask}^{\uparrow}, \overline{\mathsf{mask}}^{\downarrow}, \overline{\mathsf{mask}}^{\uparrow}, \mathsf{LT}^{\downarrow}, \mathsf{LT}^{\uparrow}, \overline{\mathsf{LT}}^{\downarrow}, \overline{\mathsf{LT}}^{\uparrow}$. *The commitments to actual masks here also corresponds to the commitment to randomness that the verifier receives at the beginning of the protocol.*
3. **Checking Well-Formedness of Packed Shares:** As before, invoke instances of the sub-protocol from Figure 4 on the above encodings to ensure consistency of packed secret shares of masks.
4. **Checking Validity of Masks:** Invokes instances of the sub-protocol from Figure 6 on the above encodings of masks and the respective ranges to ensure that the masks were sampled from appropriate ranges.

- **MPC-in-the-head (for the online phase):** Using the above shares of data, correlated randomness, and masks for non-arithmetic computations, the prover runs the online phase of MPC protocol from Figure 3 in its head. It then computes hash-based commitments to the views of each virtual party in this MPC execution and sends these n commitments to the verifier. Here, the view of each party comprises of all the shares/values received and sent by the parties. As discussed in the overview, because of the massive size, we do not include the shares of data in these views.

  In order to convince the verifier that the output of this MPC is the model that the verifier was holding a commitment to, the prover sends the following additional information: for each $i \in [n]$ it sends a commitment to a randomized polynomial encoding of the packed secret shares of $\mathbf{w}_{N/B,0}, \ldots, \mathbf{w}_{N/B,D}$ held by party

P_i. It also invokes Figure 4 on these as well as the commitments to the actual values $\mathbf{w}_{N/B,0}, \ldots, \mathbf{w}_{N/B,D}$ that the verifier was given at the beginning of the protocol.[13]

- **Sample Random Subset of Parties:** Verifier samples random field elements $r, s \in \mathbb{F}$ and a random $(t-\ell)$-sized subset $T \subset [n]$ of the parties (which includes $\mathsf{P}_{\mathsf{leader}}$) and sends r, s, T to the prover.

- **Opening Views:** Prover opens the views of parties in subset T to the verifier, including the randomness used by the prover to create polynomial commitments the shares of correlated randomness and masks held by the virtual parties in T (note that the prover does not send shares of the data or the randomness used to compute commitments to data shares). For data shares, the prover additionally sends the following for each $i \in T$:

  – *For Step I(1):*

    1. Let $\mathsf{weights}^{(i)}(X)$ be a polynomial encoding of a ND/B-length vector
    $$\left(r^1 [\![\mathbf{w}_{0,1}]\!]^{\mathsf{P}_i}, \ldots, r^1 [\![\mathbf{w}_{0,D}]\!]^{\mathsf{P}_i}, \ldots, r^{\frac{N}{B}} \left[\!\!\left[\mathbf{w}_{\frac{N}{B}-1,1}\right]\!\!\right]^{\mathsf{P}_i}, \ldots, r^{\frac{N}{B}} \left[\!\!\left[\mathbf{w}_{\frac{N}{B}-1,D}\right]\!\!\right]^{\mathsf{P}_i}\right)$$
    2. For each $u \in [B/\ell]$, let $z_u^{(i)}(X)$ be a polynomial encoding of a N/B-length vector $\left(r^1 (\mathbf{z}_1)^{(u)}, \ldots, r^{N/B} (\mathbf{z}_{N/B})^{(u)}\right)$. Let $z^{(i)}(X) = \sum_{u \in [B/\ell]} s^u \cdot z_u^{(i)}(X)$
    3. For each $u \in [B/\ell]$, let $\mathsf{alpha}_u^{(i)}(X)$ be a polynomial encoding of a N/B-length vector $\left(r^1 (\boldsymbol{\alpha}_1)^{(u)}, \ldots, r^{N/B} (\boldsymbol{\alpha}_{N/B})^{(u)}\right)$. Let $\mathsf{alpha}^{(i)}(X) = \sum_{u \in [B/\ell]} s^u \cdot \mathsf{alpha}_u^{(i)}(X)$
    4. Let $\mathsf{col}^{(i)}(X) = \sum_{u \in [B/\ell]} s^u \cdot \mathsf{col}_u^{(i)}(X)$
    5. **Sumcheck:** The prover proves the following relation using sumcheck: *sum of evaluations of* $z^{(i)}(X)$*-sum of evaluations of (*$\mathsf{alpha}^{(i)}(X)$ *on* N/B^{th} *roots of unity) - (sum of evaluations of* $\mathsf{col}^{(i)}(X) \cdot \mathsf{weights}^{(i)}(X)$ *on* ND/B^{th} *roots of unity) is 0*

  – *For Step IV(1):* The prover performs similar computations for Step IV(1) w.r.t. to the $\mathsf{row}_u^{(i)}$ polynomials and invokes a sumcheck in a similar manner at the end of it.

- **Checking Consistency of Views:** The verifier performs the following local checks for each $i \in T$:

  1. *Consistent Polynomial Commitments:* Check if the polynomial commitments to shares of correlated randomness and masks are consistent with the information received from the prover in the previous step.
  2. *Correctness of Model:* Run the verification checks from Figure 4 using the information provided by the prover about the shares of the final model (or weight vector). This allows him to verify that the this proof verifies against the commitment to the final model that the verifier was holding on to.
  3. *Sumchecks for Steps I(1) and III(1)* Verify if the sumcheck proofs that the prover provides for these steps verify.
  4. *All other steps:* For each $j \in [N/B]$, it checks if all the other views of opened parties are consistent with the description of the MPC protocol from Figure 3.

If all the above checks verify, the verifier outputs accept, else it outputs reject.

[13]We assume that the polynomial commitments to these actual weight values are computed in a similar manner as all other commitments to random values in previous steps.

Theorem 1. *The above protocol is a zero-knowledge proof of training (as defined in Section 3) for logistic regression.*

Proof of Theorem 1. As prescribed by the definition of proof of training, in our construction, the commitment to the data is simply the encoding of the data, and the commitment to the randomness are the encoding of the permutation polynomials (which defines how the data are shuffled between different iterations) and the polynomial commitment of the truncation masks (which defines the randomness used in the probabilistic truncation).

Correctness is straightforward. Soundness follows from the soundness of the MPC-in-the-head and the subroutines. First, the plain MPC-in-the-head guarantees a (statistical) soundness of $(1 - t_r/n)^{t_p}$. Recall that the robustness parameter of the MPC protocol is $t_r = n - 2t$, and the privacy parameter is $t_p = t - \ell$. For our setting of $t \approx n/4$ and $\ell \approx n/8$, the (statistical) soundness of the MPC protocol is $2^{-n/8}$, exponential in n. Next, additional soundness errors are introduced when we do not reveal the view in the clear but give succinct proofs for it using our subroutines. Since all of our subroutines have a negligible soundness error, the overall soundness of our protocol is $\mathsf{negl}(\lambda)$. The zero-knowledge property of our protocol follows from the zero-knowledge property of the MPC-in-the-head approach and the subroutines. The simulator may simulate the view of the MPC-in-the-head part first, which gives the statement of the subroutines. Next, it invokes the simulator of the subroutines to simulate the entire view of the verifier. □

6.3 Extension: zkPoT for Other Models

As discussed earlier, MPC-in-the-head-based zero-knowledge proof systems typically yield proofs whose size is linear in the size of the computation. A careful reader might have observed that for reducing the proof size of our MPC-in-the-head-based zero-knowledge proof, (amongst other optimizations,) we crucially leverage two main properties of logistic regression in the above protocol:

- *All operations except for the activation function are simple addition/subtraction and matrix multiplication:* Addition/subtraction is very efficient to implement inside an MPC without additional communication. Moreover, matrix multiplication essentially involves computing inner products, which can be computed inside an MPC with minimal communication (as discussed in Section 4.1, this is essentially what allows us to design a zkPoT where the proof size is independent of the number of features in the data set). Finally, since these addition/subtraction and matrix multiplication operations are repeated several times, we can check that they were all computed honestly in a "single-shot" using techniques from the zk-SNARKs literature.

- *The activation function, which is the only non-linear operation can be approximated using a piecewise-linear function without significant loss in accuracy [MR18]:* Piecewise-linear function is essentially a "selection function" that can also be computed quite efficiently using existing MPC techniques (see Step 2 in Figure 3).

As one might expect, the above two properties are not unique to logistic regression. Indeed, apart from a non-linear activation function, all other operations in the training procedure for more sophisticated models such as neural networks, are simple addition/subtraction and matrix multiplication operations. Moreover, it has been shown in prior works [CBD15, MZ17, CDNF+21, EMMZ20] that the activation function in such models can be approximated using a piecewise-linear function without significantly affecting accuracy.

As a result, our techniques can also be extended to design zkPoT for such models, provided their activation function can be safely approximated using a piece-wise linear function. However, since the training times in these models is already quite long, we will have to introduce other algorithmic (and systems) optimizations in order to obtain a "truly" efficient solution for larger models. We leave this for future work.

7 Implementation and Evaluation

We implemented a prototype of our Proof of Training protocol for logistic regression in Rust, which can be found at https://github.com/guruvamsi-policharla/zkpot. We rely on the `winterfell` library[14] for field arithmetic and implement our own library for packed secret sharing. We then extend the implementation of FRI (Fast Reed-Solomon Interactive Oracle Proof of Proximity) in `winterfell` to build various polynomial arguments such as a commitment scheme and the sumcheck protocol and its variants. Finally, we use these core building blocks to implement our MPC-in-the-head (MPCitH) style zero-knowledge proof. Our code is written in a modular fashion, allowing it to be easily used and extended for other applications as well.

Setup. Our experiments were run on an N2 GCP instance with 512 GB of RAM, in single-threaded mode. We emphasize that the high RAM requirement is an artifact of our implementation and is not necessary. In fact, if our implementation is upgraded to a streaming-friendly version,[15] we expect that it can be run on consumer-grade laptops. All arithmetic is implemented over a 128-bit ($p = 2^{128} - 45 * 2^{40} + 1$) prime field. The larger field size is required by the secure truncation protocol for fixed-point arithmetic(see Section 2.4). We expect that finding a way to avoid this and instead using a 64-bit field would further speed up the protocol and reduce proof size. For the MPCitH, we use 512 parties with a packing factor of 64 ($n/8$); for FRI, we use a low-degree-extension with blowup factor 2 and use 95 queries combined with 20 bits of grinding for 115 bits of security [Sta21].

Three Phases. We divide the benchmarks of our protocol into three phases to better understand the bottlenecks in different parts of the protocol.

- **Offline phase.** We categorize the offline phase as any computation that can be done before any data is made available to the prover, such as setting up randomness needed for the MPCitH. We find this to be a meaningful separation because the prover can pre-compute these offline proofs during system idle time.

- **Data phase.** We demarcate this step by any work done that is oblivious to the model training. This includes consistency checks on the data that is secret shared amongst parties in MPCitH (see Section 6).

- **Online phase.** Finally, the online phase refers to any computation related to the actual training of the model *after* the offline phase and data checks have been completed.

We believe that the above separation highlights both the advantages (a very fast and efficient online phase) and limitations of our approach (an expensive pre-processing phase) as is typical in secure multi-party computation. We focus on benchmarking the overall proof size and the prover/verifier time for the data and online phase. The online phase can further be divided into two main categories of checks:

[14] https://github.com/facebook/winterfell

[15] We discuss later in the section how this can be done.

- The first type are what we refer to as *cryptographic* checks, which involve cryptographic objects such as hash functions. These are typically slower but offer very powerful compression in proof sizes and verifier time. In our concrete instantiation, this is primarily the hash-based FRI proofs.
- The second type are *information-theoretic* checks, which only involve algebraic checks and are very fast in comparison but result in larger sizes. These are mainly the algebraic checks from the MPCitH.

Our main strategy is to use the former to compress very large parts of the witness, such as the training data, at the cost of some additional work to the prover, but at the same time, we try to maximize the use of the latter as it allows us to greatly minimize the *cryptographic overhead*, thereby striking a delicate balance between the prover/verifier work and the size of proofs.

System Design. The overall proof contains many moving parts and in our implementation, it was vital to design modular protocols that can be combined easily with each other. We first implemented a Packed Secret Sharing scheme, which works over any generic field implemented in the `winterfell` library. Next, we observe that it is sufficient to open all polynomials in the online phase (resp., offline phase), at the same random point. When this is the case, FRI openings can be batched together where instead of producing an individual opening proof for each polynomial, the polynomials can first be *folded* and then a single opening proof can be produced for the folded polynomial [Hab22]. This results in several advantages as hashing in the FRI proofs becomes a major bottleneck at the scale in which we operate and the overhead of FRI polynomial openings is no longer multiplicative and instead additive. However, this complicates our system design as we now have to *gather* polynomials from various subprotocols and prove them in one final step. We handle this by implementing a `batch-prover` struct that can be passed to various subprotocols, which can then *append* their polynomials by inserting them into a key-value store. The batch-prover is then finalized exactly once at the end of the proof creation. We note that although our implementation is not fully optimized, it is still quite performant and many of our modules can be used in a black-box manner.

Streaming Friendliness. As mentioned earlier, our design is naturally streaming friendly and as a result, we expect that our protocols can also be run on commodity laptops, even for large datasets (in the order of 10s of GBs). In the offline and data checks phases, the prover computation essentially involves embedding random values or data inside polynomials and checking if these values satisfy some requisite correlations by polynomial-based techniques (Section 5). While protocol from Section 6.2 suggests embedding all the values associated with a certain check into a *single* polynomial, we note that this is not strictly necessary. We can instead embed these values into *multiple* smaller polynomials (depending on the available memory) and perform the requisite checks on each of them separately. As expected, this will result in slightly larger proof sizes. In the online phase, the MPCitH protocol is already streaming-friendly and the cryptographic checks can be handled in a manner similar to the offline and data checks phase.

Packed Secret Sharing. We note that implementing packed secret sharing such that sharing and reconstruction both take place in $O(n \log n)$ time using FFTs is somewhat non-trivial and can be sub-optimal if not done carefully. [16] We present details here for completeness. The challenge over plain Shamir-secret sharing, stems from the fact that there are many secrets that need to be shared, which must be stored as evaluations of the polynomial over a smooth domain (for efficiency in various other parts of the protocol). Simultaneously, the shares must also be stored on a smooth domain as we also want to be able to recover

[16]For example the implementation in https://github.com/snipsco/rust-threshold-secret-sharing uses an FFT for powers of 3 which is slower than powers of 2 as benchmarked in https://medium.com/snips-ai/optimizing-threshold-secret-sharing-c877901231e5.

secrets in quasi-linear time. The strategy is to set one of these domains (say the share domain) to be the n-th roots of unity if there are n parties. Next, let ℓ be the number of secrets packed in each polynomial and t_p be the number of corrupt parties. Then, we set the secrets domain to be a *coset* of the $\ell + t_p + 1$-th roots of unity. First note that the two domains do not have any elements in common by the definition of a coset. We now rely on the fact that FFTs over cosets of a smooth domain can also be computed in quasi-linear time and thus to compute shares we simply use the coset-iFFT to obtain coefficients and compute an FFT over the n-th roots of unity. Our implementation can be found in the `secret-sharing` crate of our zkPoT library.

7.1 Benchmarks

We report the performance in each of the three phases (online, data, offline) separately. The total proof size can be estimated by simply adding up the proof size in individual phases. We also provide benchmarks for prover/verifier time in the data checks and online phase which can be added up to determine the time taken to prove/verify data dependent parts of the protocol. Although the cost of creating proofs of training may seem very large we note that a) we are able to handle large data sets that are used in the real world (10s of GB). For instance the largest data set size used in Kaggle competitions is 82 GB [17] and b) our implementation is far from optimal and further improvements can significantly reduce the cryptographic overhead. In the tables that follow, N denotes the dataset size, D the dimension of each entry and B the batch size of the mini-batch gradient descent.

Online proof size. As discussed earlier, the online phase can be separated into two parts – information-theoretic and cryptographic. For the information-theoretic checks, our proof size and prover/verifier time grow linearly with the number of batches given that it is an MPCitH proof. For the cryptographic checks, since we use FRI, the proof size and verifier time only grow poly-logarithmically and amount to < 5 MB in total, even for the largest parameters that we benchmark. In Table 1 and Table 2, we provide micro-benchmarks for the size and prover (verifier) time, respectively, of MPCitH proofs generated in each epoch. These can be used to easily estimate the online proof size and timings for any dataset size.

D \ B	128	256	512	1024
128	95.7	171	321.7	623
256	116	191.4	342	643.3
512	156.7	232.1	382.7	684
1024	238.1	313.5	464.1	765.4

Table 1: Proof size (in KB) per batch of information-theoretic checks in the online phase as the dimension and batch size are varied.

While the information-theoretic checks form the biggest chunk of the final proof of training, the cryptographic checks based on FRI take the longest to prove. We benchmark these checks for varying data set sizes and dimensions to understand the scaling behavior. In Table 3 we provide benchmarks for prover (verifier) times.

Data Set Checks. We next benchmark the prover (verifier) time to ensure that the commitments to the data and the shares of data are correctly related and second that the commitments to row and column

[17] https://www.kaggle.com/code/benhamner/competitions-with-largest-datasets

D \ B	128	256	512	1024
128	15.8 (5.2)	29.3 (10.5)	55.9 (20.9)	116.5 (42.4)
256	31.7 (5.7)	51.2 (10.8)	102.4 (21.4)	188.1 (43.1)
512	60.0 (6.3)	104.3 (11.5)	181.5 (22.1)	347.4 (44.1)
1024	117.1 (7.6)	168.9 (12.8)	298.2 (23.2)	607.1 (45.5)

Table 2: Prover (verifier) time (in ms) per batch of information-theoretic checks in the online phase as the dimension and batch size are varied.

N \ D	128	256	512	1024
2^{12}	1.1 (0.17)	1.8 (0.18)	3.1 (0.20)	5.8 (0.26)
2^{14}	5.1 (0.67)	7.9 (0.73)	13.9 (0.85)	26.0 (1.1)
2^{16}	22.52 (2.77)	35.1 (3.01)	63.3 (3.55)	117.69 (4.69)
2^{18}	104.5 (11.45)	160.2 (12.52)	274.8 (14.89)	521.1 (19.39)

Table 3: Prover (verifier) time (in s) of cryptographic checks in the online phase as the dimension and data-set size are varied with a fixed batch size of 256.

encodings have been computed correctly (see Section 6) In Table 4, we provide timings for varying data set sizes and dimensions. Since these are just FRI proofs, again owing to the fact that we can provide short batch openings of polynomials, these only cost ≈ 5 megabytes.

Offline proof size. We did not implement the offline phase and instead estimate the cost of the offline phase involving the preprocessing and randomness checks. At a high-level the offline phase consists solely of FRI proofs resulting in small proofs and verifier time but an expensive prover time. However, these can be done before the data is even available and hence pre-processed and stored for when training needs to be done, at which point, the prover only needs to run the online phase and data checks. To estimate the proof size, we use the following strategy. We first write a script to measure the number of polynomials that will need to be opened in the offline phase[18] and find this to be less than 86000 polynomials. In comparison to the queries made by the verifier on these (very large number of) polynomials ($<$ 140 MB), the FRI proof size is quite small ($<$ 500 KB). [19]

Cryptographic Overhead. To put our results in context, we compare the training time to proof genera-

[18]The number of polynomials is independent of various parameters of the machine learning such as batch size, dimensions or data set size.

[19]We arrive at $< 140MB$ because we open each polynomial at 95 points, each of which is a 128-bit field element.

N \ D	128	256	512	1024
2^{12}	4.2 (0.033)	8.6 (0.035)	14.3 (0.042)	39.8 (0.059)
2^{14}	17.7 (0.044)	38.1 (0.059)	85.1 (0.093)	178.0 (0.154)
2^{16}	83.8 (0.099)	173.3 (0.157)	383.8 (0.279)	829.7 (0.532)
2^{18}	387.3 (0.308)	817.5 (0.560)	1655.5 (1.182)	3690.2 (2.474)

Table 4: Prover (verifier) time (in s) of data checks as the dimension and data-set size are varied with a fixed batch size of 256.

tion. If we consider the task of training on a data set with 2^{18} entries, 1024 dimensions for each entry, and a batch size 1024, the online proof size is approximately 200 MB, resulting in a total proof size of $\approx$ 350 MB including proofs for the data checks and offline phase. Although this may seem large in comparison to zkSNARKs, we note that the proof size is $<$ 10% of the input dataset (4 GB). In comparison to previously proposed *non-private* solutions [PMSW18], although ours is computationally more expensive, we provide formal cryptographic guarantees of soundness against malicious provers, privacy of the model and training, and in addition our proof sizes are sub-linear in the training data, whereas prior work [PMSW18] needed to send the verifier the *entire* training dataset.

We next turn to estimate the computational overhead on top of training a model to produce a proof of training. To do so, we first implemented a logistic regression training to benchmark the time taken to do plain training. We actually implement this for two different data types. The first is using rust's native `f64` datatype, which handles floating point arithmetic and we observed that training on the above-mentioned dataset took approximately 1 s. However, the `f64` arithmetic is highly optimized and is much faster than arithmetic over the 128-bit field that we use. We also estimated the cost of training over the 128-bit field and found that even without the overhead of fixed point arithmetic and completely ignoring the activation function, training took approximately 11.5 s.

Finally, we benchmarked the time taken to produce a proof of training and found that the online phase took 518 s and data checks took close to one hour. We note that comparing the overhead of generating a proof of training against both the above times is *meaningful* as the former gives the overhead on top of optimized training algorithms, whereas the latter provides insight into how much overhead is introduced by our techniques beyond the seemingly inherent overhead introduced by moving to larger finite fields that are not natively supported by modern CPU architectures. The online phase has an overhead of $\approx 518\times$ and $\approx 45\times$ respectively, whereas the overhead of all computation excluding preprocessing (data checks + online phase) is $\approx 4200\times$ and $\approx 366\times$ respectively.

Although this may seem large, we emphasize that a) proof generation in all existing zero-knowledge (zk) proofs is slower than the time to do the actual computation and b) our implementation can be further optimized. In zk-SNARKs, this overhead is significantly higher. For instance, the fastest zk-proof of inference [LXZ21b] is approximately 1500× slower than the inference-time (this is a highly optimized zk-SNARK for a specific application, the overhead incurred by general zk-SNARKs is much higher.). Moreover, known constructions of zk-SNARKs do not scale well for large computations because they are not streaming-friendly. A prior work [BFH+20] ran out of memory when attempting to prove that a *linear* model was trained correctly on a dataset of size 2000.

Crucially, we note that there is no fundamental barrier in terms of memory requirements for how large a model can be proved using our techniques, as our algorithm is streaming-friendly and can be adapted to the needs of available hardware. We believe that MPC-in-the-Head, when carefully combined with techniques from the succinct proofs literature, offers a promising path for cryptographic proofs of training.

Acknowledgement

The first, sixth, and seventh authors are supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, and Visa Inc and the BAIR Commons Meta Fund. The third author is supported by Air Force Grant FA9550-18-1-0166, the National Science Foundation (NSF) Grants CCF-FMitF-1836978, IIS-2008559, SaTC-Frontiers-1804648, CCF-2046710, CCF-1652140, and 2039445, and ARO grant number W911NF-17-1-0405, and DARPA-GARD problem under agreement number 885000. The fifth author was supported by NSF grants CCF-1910681 and CNS1936799. The sixth author is also supported by the AI Policy Hub Fellowship from the UC Berkeley Center for Long-Term Cybersecurity. We thank Deevashwer Rathee for useful discussions on fixed point arithmetic.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017. 6

[AS19] Abdelrahaman Aly and Nigel P. Smart. Benchmarking privacy preserving scientific operations. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 509–529. Springer, Heidelberg, June 2019. 6

[ASKG19] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: Two-party secure neural network training and prediction. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1231–1247. ACM Press, November 2019. 9

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018. 26

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018. 14

[BCC$^+$17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017. 4, 5

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. 5

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 427–457. Springer, Heidelberg, May / June 2022. 7

[BCR$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019. 14, 15

[BFH$^+$20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2025–2038. ACM Press, November 2020. 5, 6, 9, 39

[BGJK21] Gabrielle Beck, Aarushi Goel, Abhishek Jain, and Gabriel Kaptchuk. Order-C secure multiparty computation for highly repetitive circuits. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 663–693. Springer, Heidelberg, October 2021. 7, 19

[BHK12] Florian Böhl, Dennis Hofheinz, and Daniel Kraschewski. On definitions of selective opening security. In *Public Key Cryptography–PKC 2012: 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings 15*, pages 522–539. Springer, 2012. 3

[BHR$^+$20] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 168–197. Springer, Heidelberg, November 2020. 7

[BHR$^+$21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 123–152, Virtual Event, August 2021. Springer, Heidelberg. 7

[blo22a] 2022. https://worldcoin.org/blog/engineering/intro-to-zkml. 9, 17

[blo22b] 2022. https://github.com/lyronctk/zator/tree/main. 9

[CBD15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3123–3131, 2015. 8, 34

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 234–252. Springer, Heidelberg, December 2008. 26

[CDG$^+$17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1825–1842. ACM Press, October / November 2017. 6

[CDG18] Sam Corbett-Davies and Sharad Goel. The measure and mismeasure of fairness: A critical review of fair machine learning. *arXiv preprint arXiv:1808.00023*, 2018. 9

[CDNF$^+$21] Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Alberto Nannarelli, Marco Re, and Sergio Spanò. A pseudo-softmax function for hardware-based high speed image classification. *Scientific reports*, 11(1):15307, 2021. 8, 34

[CGR$^+$19] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. Ezpc: Programmable and efficient secure two-party computation for machine learning. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 496–511, 2019. 9

[CPR18] Valerie Chen, Valerio Pastro, and Mariana Raykova. Secure computation for machine learning with spdz. *Workshop on Privacy Preserving Machine Learning at NeurIPS*, 2018. 6

[CRS20] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4PC framework for privacy preserving machine learning. In *NDSS 2020*. The Internet Society, February 2020. 9

[CS10] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In Radu Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 35–50. Springer, Heidelberg, January 2010. 7, 12

[EGPS22] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. TurboPack: Honest majority MPC with constant online communication. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 951–964. ACM Press, November 2022. 7

[EMMZ20] Alessandro Epasto, Mohammad Mahdian, Vahab S. Mirrokni, and Emmanouil Zampetakis. Optimal approximation - smoothness tradeoffs for soft-max functions. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 8, 34

[exa] https://www.jchs.harvard.edu/blog/high-income-black-homeowners-receive-higher-interest-rates-low-income-white-homeowners. 3

[FJT$^+$22] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. On the fundamental limits of formally (dis) proving robustness in proof-of-learning. *arXiv preprint arXiv:2208.03567*, 2022. 9

[FQZ$^+$21] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. ZEN: An optimizing compiler for verifiable, zero-knowledge neural network inferences. Cryptology ePrint Archive, Report 2021/087, 2021. https://eprint.iacr.org/2021/087. 3, 5, 9

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992. 7, 10, 19

[GHH$^+$23] Matthew Green, Mathias Hall-Andersen, Eric Hennenfent, Gabriel Kaptchuk, Benjamin Perez, and Gijs Van Laer. Efficient proofs of software exploitability for real-world processors. *Proc. Priv. Enhancing Technol.*, 2023(1):627–640, 2023. 5

[GJJZ22] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. Succinct zero knowledge for floating point computations. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1203–1216. ACM Press, November 2022. 9

[GKVZ22] Shafi Goldwasser, Michael P Kim, Vinod Vaikuntanathan, and Or Zamir. Planting undetectable backdoors in machine learning models. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 931–942. IEEE, 2022. 18

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016. 6

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985. 3

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. 18

[GPS22] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2022. 7

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. 5

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953. 15, 16

[Hab22] Ulrich Haböck. A summary on the FRI low degree test. Cryptology ePrint Archive, Report 2022/1216, 2022. https://eprint.iacr.org/2022/1216. 36

[Hal15] Jonathan I. Hall. Notes on coding theory, 2015. https://users.math.msu.edu/users/halljo/classes/codenotes/GRS.pdf. 10

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007. 4, 6, 11, 12, 18

[JYCC$^+$21] Hengrui Jia, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1039–1056. IEEE, 2021. 9

[KCC23] Zhifeng Kong, Amrita Roy Chowdhury, and Kamalika Chaudhuri. Can membership inferencing be refuted? *arXiv preprint arXiv:2303.03648*, 2023. 9

[KGC$^+$18] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 1353–1370. USENIX Association, August 2018. 5

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992. 4

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018. 6

[KPV22] Assimakis A. Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitments. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1725–1737. ACM Press, November 2022. 13

[KVH$^+$21] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 4961–4973, 2021. 6, 9

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 619–631. ACM Press, October / November 2017. 6

[LKKO20] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network. Cryptology ePrint Archive, Report 2020/584, 2020. https://eprint.iacr.org/2020/584. 3, 5, 9

[LXZ21a] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2968–2985. ACM Press, November 2021. 3, 5, 9

[LXZ21b] Tianyi Liu, Xiang Xie, and Yupeng Zhang. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2968–2985, 2021. 5, 39

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. 4, 5

[MR18] Payman Mohassel and Peter Rindal. ABY^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 35–52. ACM Press, October 2018. 8, 9, 11, 34

[MWW22] Haitham HM Mahmoud, Wenyan Wu, and Yonghao Wang. Proof of learning: Two novel consensus mechanisms for data validation using blockchain technology in water distribution system. In *2022 27th International Conference on Automation and Computing (ICAC)*, pages 1–5. IEEE, 2022. 9

[MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society Press, May 2017. 8, 9, 34

[Nao90] Moni Naor. Bit commitment using pseudo-randomness. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 128–136. Springer, Heidelberg, August 1990. 3

[ND21] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. 3

[nyt] Can We No Longer Believe Anything We See? — nytimes.com. https://www.nytimes.com/2023/04/08/business/media/ai-generated-images.html. [Accessed 09-08-2023]. 3

[PMSW18] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018. 9, 39

[PSSY21] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 2165–2182. USENIX Association, August 2021. 9

[RBS$^+$22] Deevashwer Rathee, Anwesh Bhattacharya, Rahul Sharma, Divya Gupta, Nishanth Chandran, and Aseem Rastogi. SecFloat: Accurate floating-point meets secure 2-party computation. In *2022 IEEE Symposium on Security and Privacy*, pages 576–595. IEEE Computer Society Press, May 2022. 9

[RKH$^+$21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 5

[RPX$^+$22] Deevashwer Rathee, Guru Vamsi Policharla, Tiancheng Xie, Ryan Cottone, and Dawn Song. Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi. Cryptology ePrint Archive, Paper 2022/1286, 2022. https://eprint.iacr.org/2022/1286. 5

[RRG$^+$21] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. SiRnn: A math library for secure RNN inference. In *2021 IEEE Symposium on Security and Privacy*, pages 1003–1020. IEEE Computer Society Press, May 2021. 6

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. 10

[Sta21] StarkWare. ethSTARK documentation. Cryptology ePrint Archive, Report 2021/582, 2021. https://eprint.iacr.org/2021/582. 14, 35

[SVP+12] Srinath T. V. Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In Tadayoshi Kohno, editor, *USENIX Security 2012*, pages 253–268. USENIX Association, August 2012. 9

[TJSP22] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4007–4022, 2022. 9

[TMS+23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 3

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 601–618, USA, 2016. USENIX Association. 3

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *PoPETs*, 2019(3):26–49, July 2019. 9

[WYX+21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 501–518. USENIX Association, August 2021. 9

[Yam22] Roman Yampolskiy. Unownability of ai: Why legal ownership of artificial intelligence is hard. 2022. 9

[ZFZS20] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2039–2053. ACM Press, November 2020. 5

[ZkR21] ZkRollups. An incomplete guide to rollups. https://vitalik.ca/general/2021/01/ 05/rollup.html, 2021. 5