TRIP: Thresholding in Regression with Input Privacy

Chrysa Oikonomou[1,3] and Katerina Sotiraki[1,2]

[1] Archimedes/Athena RC, Greece chr.oikonomou@athenarc.gr
[2] Yale University, New Haven, CT, USA katerina.sotiraki@yale.edu
[3] National Technical University of Athens, Greece

Abstract. Secure computation allows multiple parties to jointly evaluate a function without leaking their individual inputs. An intrinsic issue with these techniques is that they do not offer any protection against parties which may contribute bad quality or even maliciously crafted data. We introduce TRIP, a protocol which protects against malicious manipulations of the input in secure computation of linear regression tasks. Linear regression is the cornerstone in many machine learning tasks, and hence creating secure protocols for this task is a crucial step towards secure machine learning. Our protocol utilizes a novel combination of techniques from secure computation, robust statistics, and differential privacy. On synthetic data, TRIP recovers the planted ground truth; on real-world datasets, its model remains close to the clean OLS baseline under up to 40% target corruption. In terms of efficiency, our protocol runs up to 250× faster than an MPC-only baseline for 10^6 samples. Even in the smallest parameter setting, TRIP is 10× faster than our baseline.

Keywords: privacy-preserving machine learning · linear regression · malicious inputs

1 Introduction

Protecting the privacy of data is crucial for maintaining compliance with legal and regulatory frameworks such as the European General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA). Despite this fact, the relentless pursuit of innovation and efficiency frequently leads organizations to prioritize model performance over privacy safeguards. The consequences of neglecting privacy safeguards are shown in recent high-profile cases. In 2023, Italy temporarily banned ChatGPT over concerns about its data handling practices [7], emphasizing the need for transparency and compliance with GDPR. In a similar case, the Federal Trade Commission (FTC) enforced a stringent action against Weight Watchers, mandating the company to delete data and algorithms developed from children's personal information [29]. These interventions underscore the serious implications of privacy violations.

In response to these growing privacy concerns, private methods have been proposed for various computations–especially related to machine learning tasks (e.g., [16, 17, 23, 36, 40, 43, 45, 59]). In particular, in the setting of collaborative training on sensitive data, multiple computing parties engage in a computation without revealing their private data. For instance, the computing parties might be hospitals which collaboratively train a machine learning model on medical data while preserving patient privacy. Privacy considerations introduce a new set of challenges in this setting, such as the inherent difficulty of verifying the quality of the used datasets. The well-established definitions of secure computation [12, 28] do not capture such adversarial manipulations of data and consider these attacks as out-of-scope. However, in real-world systems adversaries can inject malicious data, thereby compromising the reliability and robustness of deployed machine learning models.

We initiate the study of defense mechanisms against adversarial manipulations in privacy-preserving computation. In particular, we are the first to use the lens of robust statistics, which is an area dating back to the 1960s [35, 53], in secure computation. Robust statistics effectively addresses adversarial manipulations by developing techniques to detect and mitigate the impact of anomalous data. We specifically focus on adversarially robust and private protocols for linear regression tasks. Linear regression is arguably the most widely used tool in data analysis. It is characterized by its simplicity, interpretability, and efficiency and has been the paradigmatic algorithm for building privacy-preserving systems (e.g., [30, 48, 60]) and adversarially robust statistical methods (e.g., [8, 24])

The question of whether it is possible to have both privacy and robustness into a single, efficient collaborative machine learning system has remained largely unexplored. In this work, we develop and analyze a

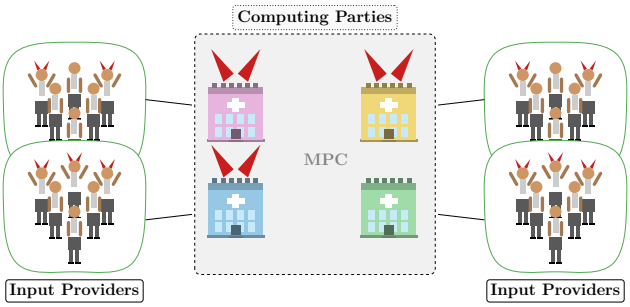


Fig. 1: Computing parties (e.g. hospitals) run an MPC to train a model, while input providers (e.g. doctors) submit raw data. Up to $N-1$ computing parties and a β fraction of input providers may be malicious.

protocol (TRIP) that protects the confidentiality of individual data entries for linear regression and simultaneously withstands corrupted datapoints. We also validate the effectiveness and efficiency of our proposed framework through extensive experiments on synthetic and real-world datasets.

In our setting, there are two categories of protocol participants: *input providers* and *computing parties*. The input providers are willing to share their input data with the computing party of their choice, but they may not trust all computing parties involved in the protocol. For example, a patient can be an input provider and may trust their local hospital with their data, but not be willing to share it with other hospitals. In this scenario, hospitals act as computing parties and collaboratively train a machine learning model while preserving data privacy. Additionally, some input providers might submit untruthful or bad quality data and all-but-one computing parties might be malicious. In the above example, this corresponds to the fact that our goal is to recover the ground truth even when hospitals use some faulty medical data. Moreover, we require that if some hospital deviates from the prescribed protocol, then the protocol aborts without leaking any private information. Figure 1 depicts our setting.

1.1 Our contributions

Our main contributions are the following:
– We construct TRIP (Thresholding in Regression with Input Privacy), guaranteeing: (1) malicious privacy with dishonest majority and (2) robustness against maliciously corrupted inputs — the first protocol to achieve both simultaneously.
– We combine Differential Privacy (DP) with Multi-Party Computation (MPC), computing on plaintext values with added noise, to achieve significant efficiency gains, while preserving formal privacy guarantees.
– We extend the filtering defense of [8] to the distributed setting via collaborative optimization, and empirically show good convergence under DP noise.
– Our implementation, available online[4], achieves up to 250× speed-up over an MPC-only baseline for 10^6 samples, 80× for varying parties or features and tolerates up to 40% malicious data.

2 Technique Overview

Problem statement. We consider high-dimensional linear regression in the setting of malicious multi-party computation (MPC), defined by:

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* + \mathbf{e}$$

where $\mathbf{w}^* \in \mathbb{R}^d$ is an unknown regression vector, $\mathbf{e}$ is sampled from a noise distribution H_e with mean 0 and variance σ_e^2, $\mathbf{X} \in \mathbb{R}^{m\times d}$, and $\mathbf{y} \in \mathbb{R}^m$. The dataset is horizontally partitioned across N parties:

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_N \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix}$$

To simplify notation, we assume each party holds n samples $(\boldsymbol{X}_i, \mathbf{y}_i) = (\mathbf{x}_{i,j}, y_{i,j})_{j=1}^n$, with $m \stackrel{\text{def}}{=} n \cdot N$. We denote the joint dataset as $(\boldsymbol{X}, \mathbf{y})$.

[4] `https://github.com/orgs/Hybrid-Private-Robust-Statistics/repositories`

Robust Statistics. We achieve robustness via the *thresholding* (or *filtering*) technique of TORRENT [8], which alternates between: (1) a *thresholding step* that identifies inlier points, and (2) a *fully corrective step* that updates the regressor on this subset. TRIP performs both steps with formal privacy guarantees.

Thresholding step. TORRENT scores each point $(\mathbf{x}_j, y_j)$, by its residual error

$$r_j = \mathbf{x}_j^\top \bar{\mathbf{w}} - y_j$$

retaining only those below the $(1-\beta)$-percentile threshold q:

$$\mathbf{s} = [\mathbb{1}(|r_j| < \mathsf{q})]_{j\in[m]} \tag{1}$$

Our main insight is that TORRENT can be adapted to a significantly more efficient MPC setting than competing robust algorithms, which require heavy operations such as exponentiation and real-value computation [46,47] or SVD [24]. In contrast, the threshold computation in TORRENT reduces to a quantile query, which we compute using the Find-Ranked-Element-MultiParty protocol $\Pi_{\texttt{qRankMPC}}$ [2]. Unlike sorting-based MPC protocols [32,33], with $\mathcal{O}(m \log m)$ communication, $\Pi_{\texttt{qRankMPC}}$ has communication complexity independent of dataset size. Moreover, prior empirical results [24,47] confirm that TORRENT achieves strong recovery guarantees in high-dimensional adversarial settings compared to classical methods [35,41].

We make two key observations that allow us to efficiently integrate $\Pi_{\texttt{qRankMPC}}$: (1) Each party must know its residuals in plaintext, hence, we reveal $\bar{\mathbf{w}}$ at each round; differential privacy (described below) mitigates the potential leakage; (2) $\Pi_{\texttt{qRankMPC}}$ does not prevent a malicious party from submitting residuals inconsistent with its private input. We address this via a tailored ZK protocol $\Pi_{\texttt{PrivSumRange}}$ (see Figure 5), which provides a proof about each party's local count of points above and below q. Finally, each party proves that it computed its participation set correctly (using eq. (1)) via $\Pi_{\texttt{PrivSumRange}}$.

Consistent Hybrid Representation. A recurring challenge is ensuring that a party's input to a ZK protocol is consistent with its input to the MPC. We address this with a *consistent hybrid representation*, a three-step process where: (1) Each party computes locally on its private data and secret-shares the result; (2) it proves the correctness of the local computation using pairwise ZK with all other parties; and (3) it proves consistency between its ZK and MPC values. This technique, summarized in Figure 3, allows us to replace expensive MPC computations with cheaper local computations and ZK proofs throughout the protocol.

Fully-corrective step. We execute the fully corrective step by evaluating a weighted least squares (WLS) regression, where the objective is to compute

$$\bar{\mathbf{w}} = \underset{\mathbf{w}}{\texttt{argmin}} \left\| \boldsymbol{S}^{1/2}(\mathbf{X}\mathbf{w} - \mathbf{y}) \right\|_2^2$$

where, $\boldsymbol{S} \stackrel{\text{def}}{=} \texttt{diag}(\mathbf{s})$ encodes the current inlier (participation) set. Since the data are horizontally partitioned, we reformulate this as a distributed constrained problem:

$$\bar{\mathbf{w}} = \underset{\mathbf{z},\mathbf{w}_i,\, i\in[N]}{\texttt{argmin}} \left(\sum_{i=1}^{N} \left\| \boldsymbol{S}_i^{1/2}(\boldsymbol{X}_i\mathbf{w}_i - \mathbf{y}_i) \right\|_2^2 \right) \quad \text{s.t. } \mathbf{w}_i - \mathbf{z} = 0 \quad \forall i \in [N]$$

where $\mathbf{w}_i$ is the local model of party $\mathcal{P}_i$, and $\mathbf{z}$ the global model. We solve this via the Alternating Direction Method of Multipliers (ADMM) [10], which naturally supports distributed computation by optimizing an augmented Lagrangian with penalty parameter $\rho > 0$ and dual variables $\mathbf{u}_i$ that capture each party's accumulated deviation from the global model. ADMM operates on the augmented Lagrangian, which removes the constraints by adding soft penalty terms to the objective:

$$\mathcal{L}_\rho(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N, \mathbf{z}, \mathbf{u}) = \sum_{i=1}^{N} \left(\left\| \boldsymbol{S}_i^{1/2}(\boldsymbol{X}_i\mathbf{w}_i - \mathbf{y}_i) \right\|_2^2 + \mathbf{u}_i^\top(\mathbf{w}_i - \mathbf{z}) + \tfrac{\rho}{2}\|\mathbf{w}_i - \mathbf{z}\|_2^2 \right)$$

where $\rho > 0$ is a penalty parameter controlling how strongly deviations of the individual models from the global model are penalized, and $\mathbf{u}_i$ are the dual variables (Lagrange multipliers) capturing the accumulated discrepancy of party $\mathcal{P}_i$'s model from $\mathbf{z}$. ADMM minimizes $\mathcal{L}_\rho$ by alternating updates over $\mathbf{w}_i$, $\mathbf{z}$, and $\mathbf{u}_i$, yielding the iterative rules in Figure 4.

Following [60], each party's contribution reduces to low-dimensional statistical summaries:

$$\boldsymbol{A}_i \equiv \left(\boldsymbol{X}_i^\top \boldsymbol{S}_i \boldsymbol{X}_i + \tfrac{\rho}{2}\boldsymbol{I}\right)^{-1} \qquad \boldsymbol{b}_i \equiv \boldsymbol{X}_i^\top \boldsymbol{S}_i \mathbf{y}_i, \tag{2}$$

which scale with d rather than n. Each party computes its summaries locally, proves correctness in ZK and using the consistent hybrid representation shares the result in MPC; the iterative ADMM updates are performed entirely in MPC at cost $\mathcal{O}(d^2)$. The full update rules appear in Figure 4.

Adding differential privacy. To avoid heavy cryptographic computations, we reveal $\bar{\mathbf{w}}$ and the threshold q at each iteration. We mitigate the potential leakage with differential privacy, which ensures these values do not meaningfully depend on any single datapoint. This DP-cryptographic approach [6,37,55] trades controlled quantified leakage for substantial efficiency gains, while preserving formal privacy guarantees — for example, [6] permits differentially private estimates of intermediate cardinalities in multi-party SQL query execution, with formal (ϵ, δ)-DP guarantees for input parties. Concretely, TRIP* perturbs $\bar{\mathbf{w}}$ using the Gaussian Mechanism [27] and releases q using the Exponential Mechanism [9], offering ϵ, δ-DP guarantees. Full details appear in Section 5.

3 Preliminaries

We provide the notation used in the paper and summarize the subprotocols we use. For a positive n, we define $[n]$ as $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. We typically use small case **bold** characters for vectors and capital **bold** characters for matrices. Let a prime field $\mathbb{F}_p$. For a vector $\mathbf{x} \in \mathbb{F}_p^d$ and $i \in [n]$, x_i denotes the i-th coordinate of $\mathbf{x}$. We use $\|\cdot\|$ for ℓ_2-norm of vectors. We denote the inner product of two vectors $\mathbf{x}_1, \mathbf{x}_2$ by $\mathbf{x}_1^\top \mathbf{x}_2$. For a matrix $\boldsymbol{A} \in \mathbb{F}_p^{n \times d}$, $a^{(i,j)}$ denotes the (i,j)-th element of $\boldsymbol{A}$. We denote by $\mathbb{1}(E)$ the indicator function of the event E. Our protocol operates in fixed-point arithmetic over a finite field $\mathbb{Z}_p$. Namely, we represent a rational number $x \in (-2^e, 2^e)$ with decimal precision f bits, i.e., $x = s \cdot 2^{-f}(\sum_{i=1}^{e+f} x_i 2^{i-1})$, where $s \in \{-1, 1\}$ is the sign and $x_i \in \{0, 1\}$, by $x \cdot 2^f \bmod p$. Note that it should hold that $p > 2^{e+f+1}$. Addition and subtraction of fixed-point values are defined in the natural way and the multiplication of two fixed-point values x, y is defined as $\lfloor xy2^f \rfloor 2^{-f}$. Signed bit decomposition of the field element $x \in \mathbb{Z}_p$ representing a fixed-point value yields bits $x_0, \ldots, x_{e+f} \in \{0, 1\}$ such that, we have $x = -x_{e+f}2^{e+f} + \sum_{i=0}^{e+f-1} x_i 2^i (\bmod\ p)$. This representation corresponds to the two's complement encoding of x.

Multi-party Computation. We utilize SPDZ [22], based on authenticated additive secret sharing — we highlight that our protocol could be realized using any arithmetic, malicious MPC. We denote authenticated additive shares of $x \in \mathbb{F}_p$ by $[\![x]\!] = \{(x_i, \gamma(x_i), \mathsf{a}_i)\}_{i=1}^N$, where x_i is $\mathcal{P}_i$'s share of x, $\gamma(x_i)$ is $\mathcal{P}_i$'s share of MAC tag on x, and a_i is $\mathcal{P}_i$'s share of the global MAC key $\mathsf{a} \stackrel{\text{def}}{=} \mathsf{a}_1 + \ldots \mathsf{a}_N \mod p$. The following relations hold: $\sum_i x_i \equiv x \pmod p$ and $\sum_i \gamma(x_i) \equiv \mathsf{a} \cdot x \pmod p$. We write $[\![x]\!]_i$ to denote the individual authenticated share held by $\mathcal{P}_i$. We assume MPC protocols for secure addition, multiplication and integer inequality/ equality comparisons of $[\![\cdot]\!]$-shared data and secure branching on $[\![\cdot]\!]$-shared conditions. For notational simplicity, $[\![z]\!] \leftarrow [\![x]\!] + [\![y]\!]$, $[\![z]\!] \leftarrow [\![x]\!] \times [\![y]\!]$, $[\![z]\!] \leftarrow [\![x]\!] \stackrel{?}{<} [\![y]\!]$, $[\![z]\!] \leftarrow [\![x]\!] \stackrel{?}{=} [\![y]\!]$, will denote secure computations. We denote $[\![x_1]\!] + \cdots + [\![x_n]\!]$ by $\sum_{i \in [n]} [\![x_i]\!]$.

Zero-Knowledge Proof. In this work, we use Quicksilver [58]. Authenticated values are subfield vector oblivious linear evaluation (sVOLE) correlations. The verifier V holds a global MAC key $\Delta \in \mathbb{F}_{p^r}$ which is sampled uniformly at random once at the beginning. The prover P knows a value $x \in \mathbb{F}_{p^r}$ and commits to it by computing a MAC tag m on that value. V is given a local key $k \in \mathbb{F}_{p^r}$ such that the MAC relation $m = k - \Delta x \in \mathbb{F}_{p^r}$ holds. That is, Pknows x and the designated Vcan authenticate it based on the MAC relation . In this context, an sVOLE share $\langle x \rangle$ means that P holds (x, m) and V holds k.

Theorem 1. *[58] constructs a zero-knowledge proof, which we denote by $\Pi_{\mathtt{polyZK}}$, for the following statement: the instance contains the multivariate polynomials $\{f_i\}_{i \in [t]}$ of degree d, the witness is $\{\mathbf{w}_i\}_{i \in [t]} \in \mathbb{F}_{p^r}^n$ and the statement is that $f_i(\mathbf{w}_i) = 0$ for all $i \in [t]$. The protocol is secure assuming Random Oracle and Learning Parity with Noise (LPN), for a security parameter κ with soundness error $(d+t)/p^r$ and it has communication complexity $nt \log p^r + d\kappa$ bits.*

Differential Privacy. Differential privacy (DP) is a property of an algorithm $\mathcal{M}$ ensuring that the participation of any single individual has only a limited effect on the released output. We use the following differentially private mechanisms: (1) The *Gaussian Mechanism* [26] $\mathcal{N}(\cdot)$ privatizes a function f by adding Gaussian noise to it. The noise depends on the *sensitivity* Δf [26] of the statistic f and the DP parameters ϵ, δ. (2) The *Exponential Mechanism* [26] $\mathcal{E}(\cdot)$ samples from the set of all possible outputs according to an exponential distribution, where outputs that are "more accurate" are sampled with higher probability. To apply this mechanism, we specify a utility function $\mathrm{u}(X, r)$, which takes an input X and a candidate r, and returns a utility score for r.

4 Our Protocol: TRIP

Symbol	Definition
m	number of total datapoints (samples)
n	number of $\mathcal{P}_i$'s samples
d	number of features of a sample
N	number of parties
$[\![x]\!]$	authenticated sharing of x
$\langle x \rangle$	sVOLE sharing of x
$\boxed{\diamond}$	$\diamond$ operation in MPC
(i)	local computation on step i
(i)	hybrid representation on step i
(i)	MPC computation on step i
(i)	ZK computation on step i
β	corruption fraction
η	additional fraction of discarded points
q	filtering rank ($q = (1 - \beta - \eta)\, m$)
q	filtering threshold value
T, t	TRIP rounds, round counter
K, k	ADMM iterations, iteration counter
$\mathcal{N}(\cdot)$	Gaussian Mechanism parametrized by ϵ, δ
$\mathcal{E}(\cdot)$	Exponential Mechanism parametrized by ϵ

Table 1: Basic Notation and Conventions

Model. We distringuish two roles. (i) *Input Providers* contribute the data the model is trained on (ii) *Computing Parties* $\{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$ jointly execute TRIP to train the model, that is, they carry out all local, ZK, MPC and hybrid steps — we assume that each pair of parties is connected via a secure and authenticated channel. For a step i: (i) denotes local computation, (i) ZK, (i) MPC, and (i) the creation of a consistent hybrid representation. All computations are over a prime field $\mathbb{F}_p$.

Threat Model. TRIP provides security against a single, *malicious* adversary, under two separate corruption bounds matching the two roles. For the computing parties, we assume a dishonest majority: *up to* $N - 1$ parties may be corrupted; this formulates privacy guarantees. For the input providers, we assume an *honest majority*: at most $\beta\%$ of the joint input may be corrupted; this formulates model robustness guarantees. The adversary is fully adaptive and can choose both the corruption locations and the specific corrupted labels, with complete knowledge of the true model and the clean datasets, and may deviate arbitrarily from the prescribed computation and manipulate inputs before and during execution. We emphasize that placing both roles under a *single* adversary is strictly stronger than using two independent ones, since it can correlate its strategy across roles: it may observe the joint input and use that view to decide how to poison the model and which computing parties to corrupt.

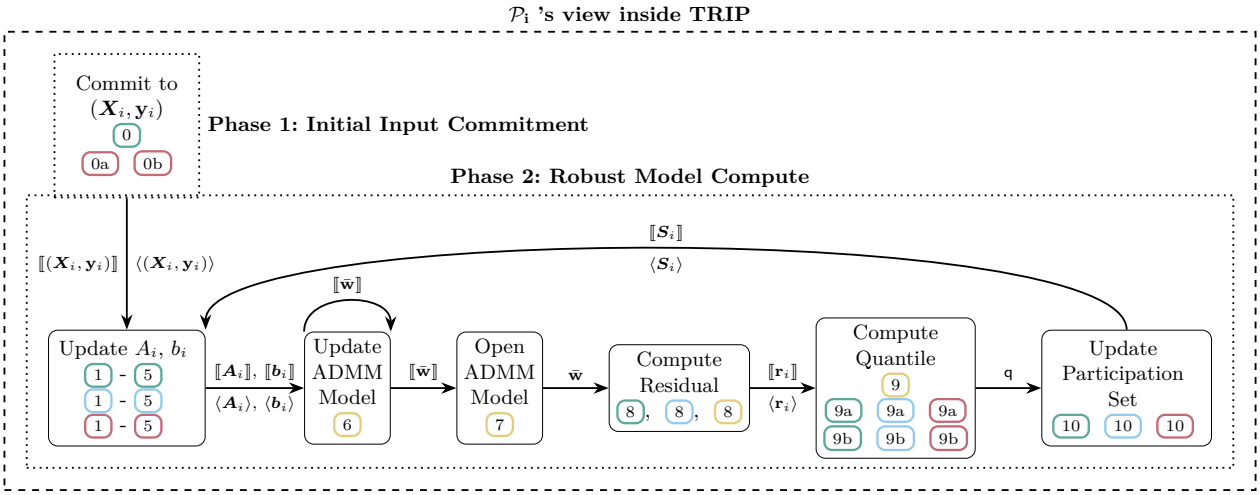


Fig. 2: MPC Protocol Workflow showing the view of party $\mathcal{P}_i$ during execution

4.1 Workflow

The protocol has two phases: (1) *Input Commitment* (runs once) and (2) *Robust Model Compute Phase* (runs iteratively for T rounds), with each round improving the model approximation towards $\mathbf{w}^*$. The workflow is illustrated in Figure 2; the formal protocol appears in Figure 6; the detailed protocol description appears in Section C; the security proof for the DP augmented protocol TRIP* appears in Section B.2.

Input Commitment Phase. Each $\mathcal{P}_i$ commits to $(\boldsymbol{X}_i, \mathbf{y}_i)$ using *Hybrid Representation* (Figure 6, (0)) via $\Pi_{\texttt{HybRep}}$ (Figure 3), obtaining consistent sharings $[\![(\boldsymbol{X}_i, \mathbf{y}_i)]\!]$ (for MPC) and $\langle(\boldsymbol{X}_i, \mathbf{y}_i)\rangle$ (for ZK). Each party, also proves its input lies within a predefined range, using Quicksilver [58, Section 4.2], ensuring no overflows, satysfying TORRENT's boundedness requirement on $\boldsymbol{X}$, and enabling calibration of DP pertubations (Section 5).

$\Pi_{\texttt{HybRep}}$: Protocol for Hybrid Representation

$\mathcal{P}_i$ commits to input x in two representations: a)SPDZ-style MAC shares and b)VOLE-style MAC shares. Each party $\mathcal{P}_{i'}$ for $i' \in [N] \setminus \{i\}$ obtains shares $[\![x]\!]_{i'}, \langle x \rangle_{i'}$.

1. **MPC Input.** Run MPC sharing among N parties so each $\mathcal{P}_{i'}$ gets $[\![x]\!]_{i'}$.
2. **VOLE Input.** $\mathcal{P}_i$ (prover) runs $N-1$ IZK protocols with each $\mathcal{P}_{i'}$ (verifier), creating $\langle x \rangle_{i'}$ for $i' \in [N] \setminus \{i\}$.
3. **Consistency Check.** $\mathcal{P}_i$ proves via $\Pi_{\texttt{CC}}$ [15] that $[\![x]\!]$, $\langle x \rangle_{i'}$ encode the same value for all $i' \in [N] \setminus \{i\}$.

Fig. 3: Hybrid Representation Protocol

Robust Model Compute Phase. In this phase, the parties execute a private and distributed version of the TORRENT for T rounds, denoted by $t \in \{1, \ldots, T\}$. Each round alternates between: (1) computing an updated global model, and (2) recomputing the participation set based on the new model and the threshold. In the following, we describe the sub-routines run inside this phase.

Input Update. Each party updates its summary statistics $(\boldsymbol{A}_i^t, \boldsymbol{b}_i^t)$ using eq. (2) (Figure 6, ((1) − (5))), over its current inlier subset $(\boldsymbol{S}_i^t \boldsymbol{X}_i,\ \boldsymbol{S}_i^t \mathbf{y}_i)$, where $\boldsymbol{S}_i^t \stackrel{\text{def}}{=} \texttt{diag}(\mathbf{s}_i^t)$ encodes the current participation set. Each party commits to these summaries consistenly across MPC and ZK (Figure 6, ((1) − (5), (1) − (5))).

Model Update and Reveal. Parties run ADMM [10] for WLS linear regression (Figure 4) on their summaries $(\boldsymbol{A}_i^t, \boldsymbol{b}_i^t)$ for K steps (e.g., $K = 5$) to obtain $[\![\bar{\mathbf{w}}^t]\!]$ (6), then reveal $\bar{\mathbf{w}}^t$ (7) (with DP noise in TRIP*, see Section 5.2).

Compute Residuals and Quantile. Parties obtain hybrid shares of each $\mathcal{P}_i$'s residual vector $\mathbf{r}_i^t \stackrel{\text{def}}{=} \boldsymbol{X}_i \cdot \bar{\mathbf{w}}^t - \mathbf{y}_i$ (Figure 6, ((8), (8), (8))). They then compute the quantile $\mathfrak{q}^t$ (Figure 6, (9)) of the absolute residuals via a

Π_{ADMM} : Distributed ADMM for WLS Regression.

Parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ with inputs $\boldsymbol{A}_i \in \mathbb{R}^{d\times d}, \boldsymbol{b}_i \in \mathbb{R}^d$, penalty $\rho > 0$, and K steps. Initialize $\mathbf{w}_i^0, \mathbf{u}_i^0, \bar{\mathbf{w}}^0 = \mathbf{0}$. For $k \in \{1, \ldots, K\}$:

1. **Indiv. Model Update** For $i \in [N]$: $[\![\mathbf{w}_i^k]\!] \leftarrow [\![\boldsymbol{A}_i]\!] \times ([\![\boldsymbol{b}_i]\!] + \frac{\rho}{2}[\![\bar{\mathbf{w}}^{k-1}]\!] - \frac{1}{2}[\![\mathbf{u}_i^{k-1}]\!])$
2. **Global Model Update.** $[\![\bar{\mathbf{w}}^k]\!] \leftarrow \frac{1}{N}\sum_{i\in[N]}[\![\mathbf{w}_i^k]\!]$
3. **Lagrange Multipliers Update** For $i \in [N]$: $[\![\mathbf{u}_i^k]\!] \leftarrow [\![\mathbf{u}_i^{k-1}]\!] + \rho\left([\![\mathbf{w}_i^k]\!] - [\![\bar{\mathbf{w}}^k]\!]\right)$

Fig. 4: Distributed ADMM for weighted least squares linear regression

modified $\Pi_{\texttt{qRankMPC}}$ [2] (see Figure 8), setting $q = (1-\beta-\eta)m$ where $\eta < 1$ is a small constant (e.g. $\eta = 0.1$) required by TORRENT to ensure slightly more than the corrupted fraction is filtered out.

Participation Set Update. Each party updates and commits to $\mathbf{s}_i^{t+1}$ (Figure 6, (10, 10)) and proves correctness via $\Pi_{\texttt{PrivSumRange}}$, (Figure 5), i.e. that $\langle \mathbf{s}_i^{t+1}\rangle_{i'} = [\langle \mathbb{1}\left(|r_{i,j}| < \mathsf{q}^t\right)\rangle_{i'}]_{j\in[n]}$(Figure 6, 10).

4.2 Modified $\Pi_{\texttt{qRankMPC}}$

A key challenge in the *Participation Set Update* is ensuring that each party's inputs to $\Pi_{\texttt{qRankMPC}}$ are consistent with its committed residual shares $[\![\mathbf{r}]\!]$. Without an explicit check, an malicious party could use arbitrary inputs rather than its true residuals. We address this by augmenting $\Pi_{\texttt{qRankMPC}}$ with $\Pi_{\texttt{PrivSumRange}}$ zero-knowledge proofs.

$\Pi_{\texttt{PrivSumRange}}$: Protocol for Private Sum of Range Checks.

P holds $\mathbf{r} \in \mathbb{F}_p^n, \mathbf{s} \in \{0,1\}^n$, and $c \in \mathbb{F}_p$. V holds $\langle \mathbf{s}\rangle, \langle \mathbf{r}\rangle$ and $\langle c\rangle$. P proves that $\mathbf{s} = [\mathbb{1}(q_1 < r_i < q_2)]_{i\in[n]}$ for public values $q_1, q_2 \in \mathbb{F}_p$ and $\mathbf{s}^\top \cdot \mathbf{1} = c$, i.e., that $\mathbf{s}$ contains c ones. Let $k \stackrel{\text{def}}{=} \lceil \log_2(p)\rceil$.

1. `Prepare Input`: P sets $[q_1', q_2'] = [q_1 + 2^{-f}, q_2 - 2^{-f}]$[a] and computes decompositions $\mathbf{b}_1, \mathbf{b}_2 \in \{0,1\}^{n\times k}$ of $q_1' - \mathbf{r}, \mathbf{r} - q_2'$ respectively. Parties create $\langle \mathbf{b}_1\rangle, \langle \mathbf{b_2}\rangle$.
2. `Perform Zero Checks` via $\Pi_{\texttt{polyZK}}$:
   (a) **k-Bit Decomposition.** Let $f(x) = x(1-x)$; for $i \in [n]$, $j \in [k]$:
   $$f(b_1^{(i,j)}) = 0 \text{ and } f(b_2^{(i,j)}) = 0.$$
   (b) **Decomposition is Correct.** Let $g_1(x_1, \ldots, x_k, x_{k+1}) = -(q_1' - x_{k+1}) + (-2^{k-1}x_k + \sum_{j=1}^{k-1} x_j 2^{j-1})$ and $g_2(x_1, \ldots, x_k, x_{k+1}) = -(x_{k+1} - q_2') + (-2^{k-1}x_k + \sum_{j=1}^{k-1} x_j 2^{j-1})$; for $i \in [n]$:
   $$g_1(b_1^{(i,1)}, \ldots, b_1^{(i,k)}, r_i) = 0 \text{ and } g_2(b_2^{(i,1)}, \ldots, b_2^{(i,k)}, r_i t) = 0$$
   (c) **Sign Bit And.** Let $h_1(x,y,z) = xy - z$; for $i \in [n]$:
   $$h_1(b_1^{(i,k)}, b_2^{(i,k)}, s_i) = 0$$
   (d) **Sum of Sign Bits.** Let $h_2(x_1, \ldots, x_n, x_{n+1}) = x_{n+1} - \sum_{i\in[n]} x_i$:
   $$h_2(s_1, \ldots, s_n, c) = 0$$

[a] P shrinks $[q_1, q_2]$ by the minimum fixed-point value 2^{-f} to prove strict inequalities.

Fig. 5: Private Sum of Range Checks (colored parts denote the sum variant).

Background: $\Pi_{\texttt{qRankMPC}}$. $\Pi_{\texttt{qRankMPC}}$ [3] securely computes the qth-ranked element of integer values in $[\alpha, \beta]$ via binary search: starting from $\mathsf{q}_1 = \frac{\alpha+\beta}{2}$, it halves the search range by comparing aggregate counts

$$\mathrm{l}_i(\mathsf{q}_l) \stackrel{\text{def}}{=} \#\{x \in D_i : x < \mathsf{q}_l\}, \ \mathrm{g}_i(\mathsf{q}_l) \stackrel{\text{def}}{=} \#\{x \in D_i : x > \mathsf{q}_l\} \tag{3}$$

until convergence. Communication complexity is $\mathcal{O}(\log_2 M)$, where $M = \beta - \alpha + 1$, which scales better than sorting-based approaches.

Our modification: Consistency via $\Pi_{\texttt{PrivSumRange}}$. After computing guess q_l^t in iteration l, each $\mathcal{P}_i$ must prove that its counts $\mathrm{l}_i(\mathsf{q}_l^t)$ and $\mathrm{g}_i(\mathsf{q}_l^t)$(computed at steps 9a, 9b and committed at steps 9a, 9b of Figure 6) were derived from its committed residuals. This is done using $\Pi_{\texttt{PrivSumRange}}$ (Figure 5) which lets a prover P show that for committed $\mathbf{x} \in \mathbb{F}_p^n$, $\mathbf{y} \in \{0,1\}^n$, and $c \in \mathbb{Z}$: $\mathbf{y} = [\mathbb{1}(x_j \in [a,b])]_{j\in[n]}$ and $c = [\mathbf{1}^n]^T \cdot \mathbf{y}$. Concretely, each $\mathcal{P}_i$ proves (steps 9a, 9b):

$$\mathbf{y}_1 = [\mathbb{1}(|r_{i,j}| < \mathsf{q}_l^t)]_{j\in[n]},\ \mathrm{l}_i(\mathsf{q}_l^t) = [\mathbf{1}^n]^T \cdot \mathbf{y}_1,\ \mathbf{y}_2 = [\mathbb{1}(|r_{i,j}| > \mathsf{q}_l^t)]_{j\in[n]},\ \mathrm{g}_i(\mathsf{q}_l^t) = [\mathbf{1}^n]^T \cdot \mathbf{y}_2,$$

where $r_{i,j}, j \in [n]$ are $\mathcal{P}_i$'s residuals committed in the previous step. [5]

Π_{TRIP}: Protocol for Private and Robust Linear Regression

Run between N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ with inputs $\boldsymbol{X}_i \in \mathbb{R}^{n\times d}$,$\mathbf{y}_i \in \mathbb{R}^n$, bounds B_x, B_y and parameters $T, K, L, \beta, \eta, \boldsymbol{E}$.

Input Commitment Phase: Each $\mathcal{P}_i$ runs $\Pi_{\texttt{HybRep}}$ (fig. 3) to commit to $(\boldsymbol{X}_i, \mathbf{y}_i)$ 0 and proves all $\mathcal{P}_{i'}$, $i' \in [N] \setminus i$ via $\Pi_{\texttt{RangeCheck}}$:

$$\|\mathbf{x}_{i,j}\|^2 < B_x^2 \ \text{0a} \qquad |y_{i,j}| < B_y \ \text{0b}$$

Robust Model Compute Phase: Executed for $t \in [T]$ rounds initialized with $\mathbf{s}_i^1 = \mathbf{1}^n$ and $\boldsymbol{S}_i^t = \texttt{diag}(\mathbf{s}_i^t)$.

1. **Input Update:** For each $i \in [N]$, $\mathcal{P}_i$ computes and commits via $\Pi_{\texttt{HybRep}}$:
$$\boldsymbol{D}_i^t \overset{\text{def}}{=} \boldsymbol{X}_i^\top \boldsymbol{S}_i^t \ 1 \quad \boldsymbol{C}_i^t \overset{\text{def}}{=} \boldsymbol{D}_i^t \boldsymbol{X}_i + \tfrac{\rho}{2}\boldsymbol{I} \ 2 \quad \boldsymbol{A}_i^t \overset{\text{def}}{=} (\boldsymbol{C}_i^t)^{-1} \ 3 \quad \boldsymbol{H}_i^t \overset{\text{def}}{=} \boldsymbol{A}_i^t \boldsymbol{C}_i^t \ 4 \quad \boldsymbol{b}_i^t \overset{\text{def}}{=} \boldsymbol{D}_i^t \mathbf{y}_i \ 5$$
$$\boldsymbol{D}_i^t \ 1 \quad \boldsymbol{C}_i^t \ 2 \quad \boldsymbol{A}_i^t \ 3 \quad \boldsymbol{H}_i^t \ 4 \quad \boldsymbol{b}_i^t \ 5$$
Then, $\mathcal{P}_i$ proves correctness to all $\mathcal{P}_{i'}$, $i' \in [N] \setminus i$ via $\Pi_{\texttt{polyZK}}$ [58]:
$$\boldsymbol{D}_i^t = \boldsymbol{X}_i^\top \boldsymbol{S}_i^t \ 1 \quad \boldsymbol{C}_i^t - \tfrac{\rho}{2}\boldsymbol{I} = \boldsymbol{D}_i^t \boldsymbol{X}_i \ 2 \quad \boldsymbol{H}_i^t = \boldsymbol{A}_i^t \boldsymbol{C}_i^t \ 4 \quad \boldsymbol{b}_i^t = \boldsymbol{D}_i^t \mathbf{y}_i \ 5$$
and via $\Pi_{\texttt{RangeCheck}}$ proves the approximate inverse property:
$$\left|\boldsymbol{H}_i^t - \boldsymbol{I}\right|_\infty < \boldsymbol{E} \ 3\ , \text{ where } \boldsymbol{E} \text{ captures the fixed point truncation error.}$$
2. **Model Update:** Parties run $\Pi_{\texttt{ADMM}}$(fig. 4) in MPC to obtain $[\![\bar{\mathbf{w}}^t]\!]$ 6 .
3. **Model Reveal:** Parties reconstruct $\bar{\mathbf{w}}^t$ 7.
4. **Participation Set Update:**
   (a) Residuals: Each $\mathcal{P}_i$ computes $\mathbf{r}_i^t \overset{\text{def}}{=} \boldsymbol{X}_i \bar{\mathbf{w}}^t - \mathbf{y}_i$ 8. The parties compute the sharing $[\![\mathbf{r}_i^t]\!] \leftarrow [\![\boldsymbol{X}_i]\!] \times \bar{\mathbf{w}}^t - [\![\mathbf{y}_i]\!]$ 8 in MPC, and $\mathcal{P}_i$ runs $\Pi_{\texttt{HybRep}}$ (fig. 3) to create a hybrid representation of $\mathbf{r}_i^t$ 8.
   (b) Quantile q^t: Parties set $q \overset{\text{def}}{=} (1-\beta-\eta)m$ and run $\Pi_{\texttt{qRankMPC}}$ in MPC 9, to compute the q-th ranked absolute residual. In each iteration $l \in [L]$, each party locally computes:
   $$\mathrm{l}_i(\mathsf{q}_l^t) \overset{\text{def}}{=} \#\left\{x \in \mathbf{r}_i : |x| < \mathsf{q}_l^t\right\} \ \text{9a} \quad \mathrm{g}_i(\mathsf{q}_l^t) \overset{\text{def}}{=} \#\left\{x \in \mathbf{r}_i : |x| > \mathsf{q}_l^t\right\} \ \text{9b}$$
   commits to both using $\Pi_{\texttt{HybRep}}$ (fig. 3) 9a, 9b, and proves correctness to all $\mathcal{P}_{i'}$, $i' \in [N] \setminus i$ via $\Pi_{\texttt{PrivSumRange}}$ (fig. 5):
   $$\sum_{j\in[n]} \mathbb{1}\left(|r_{i,j}^t| < \mathsf{q}_l^t\right) = \mathrm{l}_i(\mathsf{q}_l^t) \ \text{9a} \quad \sum_{j\in[n]} \mathbb{1}\left(|r_{i,j}^t| > \mathsf{q}_l^t\right) = \mathrm{g}_i(\mathsf{q}_l^t) \ \text{9b}$$
   (c) Participation Set: Each $\mathcal{P}_i$ computes $\mathbf{s}_i^{t+1} \overset{\text{def}}{=} [\mathbb{1}(|r_{i,j}^t| < \mathsf{q}^t)]_{j\in[n]}$ 10, commits via $\Pi_{\texttt{HybRep}}$ (fig. 3) 10, and proves correctness via $\Pi_{\texttt{PrivSumRange}}$ (fig. 5):
   $$\mathbf{s}_i^{t+1} = \left[\mathbb{1}\left(|r_{i,j}^t| < \mathsf{q}^t\right)\right]_{j\in[n]} \ \text{10}$$

Fig. 6: Main Protocol

5 TRIP*: Adding Differential Privacy to Model and Quantile Updates

A drawback of TRIP is that intermediate models and thresholds are revealed in plaintext at each iteration. Although these do not directly expose individual datapoints, an adversary may still infer sensitive information. We address this using differential privacy. TRIP* (Figure 7) modifies TRIP to release the quantiles q^t and the models $\bar{\mathbf{w}}^t$ under DP guarantees. Security follows the Ideal-Real world paradigm; the ideal functionality and the security proof appear in Section B.2. The protocol assumes access to a malicious, arithmetic MPC functionality $\mathcal{F}_{AMPC}$ [22], a ZK functionality $\mathcal{F}_{polyzk}$ [58], and the security of $\Pi^{\text{DP}}_{\texttt{qRankMPC}}$ (Figure 8).

[5] For $\mathbf{y}_2$, we prove $y_{2,j}^- = \mathbb{1}(r_{i,j} < -\mathsf{q}_l^t)$ and $y_{2,j}^+ = \mathbb{1}(r_{i,j} > \mathsf{q}_l^t)$; since $\Pi_{\texttt{PrivSumRange}}$ enforces exact semantics, these are boolean and mutually exclusive, and we set $\mathbf{y}_2 = [y_{2,j}^- + y_{2,j}^+]_{j\in[n]}$.

Π_{TRIP^*}: DP Augmented Private & Robust Linear Regression

Parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ with inputs $\boldsymbol{X}_i \in \mathbb{R}^{n \times d}$, $\mathbf{y}_i \in \mathbb{R}^{n \times 1}$, run Π_{TRIP} augmented with DP parameters $\epsilon_{\text{GM}}, \epsilon_{\text{EM}}, \delta_{\text{GM}}$:

1. Perform the Initial Input Commitment Phase from Π_{TRIP}.
2. Pre-generate $[\![\sigma_w^t]\!]$ for $t \in [T]$, where $\sigma_w^t \sim \mathcal{N}_{\mathbb{Z}}(0, 2\ln(1.25\frac{1}{\delta_{\text{GM}}})(\frac{\Delta \bar{\mathbf{w}}}{\epsilon_{\text{GM}}})^2)$ is d-dimensional discrete Gaussian noise.
3. For $t \in [T]$:
   (a) Run Robust Model Compute steps through (6).
   (b) Compute $[\![\bar{\mathbf{w}}^t_{\epsilon_{\text{GM}}}]\!] = [\![\bar{\mathbf{w}}^t]\!] + [\![\sigma_w^t]\!]$ and perform (7), (8).
   (c) Run $\Pi^{\text{DP}}_{\texttt{qRankMPC}}$ with ϵ_{EM} to obtain q^t (9) and perform checks ((9a) (9b) (9a) (9b) (9a) (9b)).
   (d) Proceed with remaining steps of Π_{TRIP}.

Fig. 7: TRIP*: Π_{TRIP} augmented with DP via the Gaussian and Exponential mechanisms; overall privacy follows by Rényi composition over all rounds.

5.1 Differentially Private q^{th}-ranked element

We present $\Pi^{\text{DP}}_{\texttt{qRankMPC}}$, a differentially private variant of $\Pi_{\texttt{qRankMPC}}$ that hides both the intermediate binary-search steps and the final q^{th}-ranked value using the exponential mechanism, , following [9]. Using fixed-point arithmetic, with values in $[-2^e, 2^e]$ and precision f, the protocol requires $L = e + f + 1$ rounds.

Utility function. In our setting, the utility of a candidate output is determined by its rank relative to the target rank q. Concretely, given dataset D with $|\mathrm{D}| = m$, let $\mathrm{l}(y) \stackrel{\text{def}}{=} \#\{x \in \mathrm{D} : x < y\}$. Rather than assigning utility per element, over the large domain $[\alpha, \beta]$, we follow $\Pi_{\texttt{qRankMPC}}$ and assign utility per subrange: for $[a, b]$ split at q into $R_1 = [a, \mathsf{q})$ and $R_2 = [\mathsf{q}, b]$,

$$\mathrm{u}(\mathrm{D}, R_1) = \begin{cases} \mathrm{l}(\mathsf{q}) - q & \text{if } \mathrm{l}(\mathsf{q}) < q \\ 0 & \text{otherwise} \end{cases} \qquad \mathrm{u}(\mathrm{D}, R_2) = \begin{cases} q - \mathrm{l}(\mathsf{q}) - 1 & \text{if } \mathrm{l}(\mathsf{q}) > q - 1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

A subrange scores 0 if it contains the target rank q, and decreases proportionally to its distance from q. One can verify that adding or removing a single element changes the utility by at most 1, giving sensitivity 1 — far smaller than the sensitivity of a naive additive mechanism over the same range.

$\Pi^{\text{DP}}_{\texttt{qRankMPC}}$ *protocol.* $\Pi^{\text{DP}}_{\texttt{qRankMPC}}$ (Figure 8) adapts $\Pi_{\texttt{qRankMPC}}$ with the exponential mechanism. Unlike $\Pi_{\texttt{qRankMPC}}$ it never early-terminates, since there is no *exact hit* case in DP, and the current candidate q is always kept in R_2. Each iteration proceeds as:

1. Set the candidate DP value of the qth-ranked element as the midpoint of current $[a, b]$: $\mathsf{q} = \lfloor a + b/2 \rfloor$.
2. Each party locally computes and secret-shares $\mathrm{l}_i(\mathsf{q})$ (as defined in Equation (3)).
3. The parties jointly compute in MPC the utility scores $\mathrm{u}(\mathrm{D}, [a, \mathsf{q}))$ and $\mathrm{u}(\mathrm{D}, [\mathsf{q}, b])$ for each subrange.
4. The parties select the next iteration's subrange in MPC with probability proportional to the exponentiated utility of each subrange.

When computing $\exp(\mathrm{u}(\mathrm{D}, R_i))$, the exponent can reach $-\mathcal{O}(m)$, falling outside the fixed-point range. We map underflowing values to zero, since extremely negative exponents correspond to negligible probabilities, enabling exponentiation via a constant number of multiplications.

5.2 Differentially Private Model Release

Apart from the threshold values, TRIP also reveals the intermediate models. We add differential privacy to the intermediate models using the standard technique from [27], which ensures approximate differential privacy by adding Gaussian noise to the final output.

Adding Gaussian Noise Directly on $\bar{\mathbf{w}}$. In contrast to [27] which adds independent noise to the summary statistics $\boldsymbol{A}$ and $\boldsymbol{b}$, proportional to their sensitivity, our proposed method adds Gaussian noise directly into $\bar{\mathbf{w}}$. This design has two key benefits: (i) it is nearly as computationally efficient as its non-DP analogue, since the parties only sample noise once per iteration, (ii) under certain assumptions (e.g., isotropic data), it yields a tighter sensitivity bound compared to [27].

$\Pi^{DP}_{qRankMPC}$: Quantile Computation in MPC with Differential Privacy

Procedure select(w_1, w_2)

i. Sample $[\![t]\!] \xleftarrow{\$} (0,1)$;
ii. Compute $[\![w_r]\!] \leftarrow ([\![w_1]\!] + [\![w_2]\!]) \times [\![t]\!]$;
iii. Set $[\![b]\!] \leftarrow [\![w_1]\!] \overset{?}{<} [\![w_r]\!]$;
iv. Open $[\![b]\!]$ and **return** $i = 1$ if $b = 0$ else $i = 2$.

Setup. Parties $\{\mathcal{P}_i\}_{i\in[N]}$ hold $\mathrm{D}_i \subseteq [\alpha, \beta]$, $m = \sum_i |\mathrm{D}_i|$, $\mathrm{D} = \biguplus_i \mathrm{D}_i$, $L = \lceil \log_2(\beta - \alpha + 1) \rceil$. Fix ϵ per round; overall privacy follows by Rényi composition (Section 5.3).
Init. Set $[a, b] \leftarrow [\alpha, \beta]$. **For** $l = 1$ **to** L**:**

1. Set $\mathsf{q} \leftarrow \lfloor (a+b)/2 \rfloor$. Define $R_1 = [a, \mathsf{q})$, $R_2 = [\mathsf{q}, b)$.
2. Each party computes locally and creates sharings $[\![\mathrm{l}_i(\mathsf{q})]\!]$ and $[\![\mathrm{g}_i(\mathsf{q})]\!]$.
3. Parties compute $[\![\mathrm{l}(\mathsf{q})]\!] \leftarrow \sum_{i\in N} [\![\mathrm{l}_i(\mathsf{q})]\!]$ and $[\![\mathrm{g}(\mathsf{q})]\!] \leftarrow \sum_{i\in N} [\![\mathrm{g}_i(\mathsf{q})]\!]$ in MPC.
4. **Sub-range selection.**

Deterministic:

$$i \leftarrow \begin{cases} 0 & [\![\mathrm{l}(\mathsf{q})]\!] \overset{?}{\leq} q-1 \ \ [\![\mathrm{g}(\mathsf{q})]\!] \overset{?}{\leq} m-q, \\ 1 & [\![\mathrm{l}(\mathsf{q})]\!] \overset{?}{\geq} q, \\ 2 & [\![\mathrm{g}(\mathsf{q})]\!] \overset{?}{\geq} m-q+1. \end{cases}$$

DP:

$$[\![w_j]\!] \leftarrow \begin{cases} [\![0]\!] & [\![\mathrm{u}(\mathrm{D}, R_j)]\!] \overset{?}{\leq} -5, \\ [\![\exp(\epsilon\, \mathrm{u}(\mathrm{D}, R_j)/2)]\!] & [\![\mathrm{u}(\mathrm{D}, R_j)]\!] \overset{?}{\geq} -5. \end{cases}$$

$$i \leftarrow \mathsf{select}([\![w_1]\!], [\![w_2]\!])$$

5. **Update interval:**

Deterministic:

$$[a, b] \leftarrow \begin{cases} \textbf{output } \mathsf{q}, & i=0, \\ [a, \mathsf{q}) & i=1, \\ (\mathsf{q}, b] & i=2. \end{cases}$$

DP:

$$[a, b] \leftarrow \begin{cases} [a, \mathsf{q}) & i=1, \\ [\mathsf{q}, b] & i=2. \end{cases}$$

DP Output. Return any value in the final interval $[a, b]$.

Fig. 8: Quantile computation in MPC; DP affects only sub-range selection via the exponential mechanism, with privacy loss composing over L rounds. annotations denote DP components.

ℓ_2-Sensitivity of intermediate models. The main challenge is bounding the sensitivity $\Delta\bar{\mathbf{w}}$ of intermediate models, to changing a single datapoint. Lemma 1 gives an upper bound for datasets whose covariance matrix has singular values of order m; Lemma 2 shows that isotropic datasets naturally satisfy this condition. For real-world datasets, where feature correlations are unknown, we adopt the technique of [27], adding independent noise to the summary statistics $\boldsymbol{A}$ and $\mathbf{b}$, at the cost of a larger privacy budget. While this incurs an additional overhead –we must add noise into $d^2 + d$ values instead of just d– as we show in Section 6.1, the utility loss remains limited when d is small and more privacy budget is available.

Sampling Discrete Gaussian Noise. To implement DP model release, parties securely sample discrete Gaussian noise following prior work (see [13, 25]). We approximate sampling from $\mathcal{N}(0, \sigma^2)$ by summing $k = 2\sigma^2$ independent $\{-1, +1\}$ bits, which by the Central Limit Theorem yields mean 0 and variance σ^2. We scale σ to match the fixed-point representation, perform the sampling, and rescale the result. Sampling in MPC uses the maliciously secure technique from [52].

5.3 TRIP*'s Privacy Budget Analysis

Using Rényi Differential Privacy (RDP) [44], which yields tighter composition guarantees, we bound the cumulative privacy loss of TRIP* (Figure 7) by composing in the RDP framework and then converting back to standard (ϵ, δ)-DP via [44, Proposition 3], resulting in an improved overall privacy budget bound. Using Proposition 1, we minimize $\epsilon(\alpha)$ over $\alpha > 1$ to obtain the tightest (ϵ, δ)-DP guarantee, determining the optimal α numerically ($\alpha^* = \sqrt{\frac{\ln(1/\delta)}{T\left(\frac{\epsilon_{\mathrm{GM}}^2}{4\ln(1.25/\delta_{\mathrm{GM}})} + \frac{L\epsilon_{\mathrm{EM}}^2}{8}\right)}} + 1$) given T, L, Δ, σ, and $\delta_{\mathrm{final}} = \delta + \delta' < 1/m$. Specifically,

we achieve $\epsilon = 2.73$, by minimizing ϵ of Equation (5) over α with $T = 5, L = 16, \epsilon_{\text{EM}} = 0.0625, \delta = 10^{-5}$. In contrast, standard composition [26] would yield $\epsilon = 10$, with the same utility.

Proposition 1. *Let T denote the total number of rounds executed by TRIP* and let L be the number of rounds of $\Pi^{\text{DP}}_{\texttt{qRankMPC}}$. Let σ be the standard deviation of the Gaussian noise added to $\bar{\mathbf{w}}$ by the Gaussian Mechanism $\mathcal{N}[\epsilon_{\text{GM}}, \delta_{\text{GM}}]$, where $(\epsilon_{\text{GM}}, \delta_{\text{GM}})$ denote the privacy parameters of the Gaussian Mechanism. Let ϵ_{EM} denote the privacy parameter of the Exponential Mechanism $\mathcal{E}[\epsilon_{\text{EM}}]$. Let $\mathcal{E}$ denote the event that the bound of Lemma 1 holds, which occurs with probability at least $1-\delta'$. On $\mathcal{E}$, the ℓ_2-sensitivity of $\bar{\mathbf{w}}$ is bounded by Δ. We condition the privacy analysis on $\mathcal{E}$; on the complement event $\mathcal{E}^c$, we upper bound the privacy loss by 1 and account for its probability in the $\delta_{final} = \delta + \delta'$. Then, for $\alpha > 1$ and any $0 < \delta < 1$, TRIP* (Figure 7) satisfies*

$$\left(T\alpha\frac{\epsilon^2_{\text{GM}}}{4\ln(1.25/\delta_{\text{GM}})} + TL\alpha\frac{\epsilon^2_{\text{EM}}}{8} + \frac{\ln(1/\delta)}{\alpha-1},\ \delta+\delta'\right)\text{-DP.} \tag{5}$$

Proof. We first calculate the Rényi DP of TRIP* using the composition theorem for Rényi DP (see Proposition 3) and then we convert Rényi-DP to DP using Proposition 4 .

We recall the Rényi DP guarantees of the Gaussian and the Exponential Mechanism. The Gaussian mechanism $\mathcal{N}[\epsilon_{\text{GM}}, \delta_{\text{GM}}]$ (Definition 4) that adds $\mathcal{N}(0, \sigma^2 I)$ noise ($\sigma = \frac{\Delta\sqrt{2\ln(1.25/\delta_{\text{GM}})}}{\epsilon_{\text{GM}}}$) to a solution with ℓ_2-sensitivity Δ satisfies $\left(\alpha, \alpha\frac{\Delta^2}{2\sigma^2}\right)$-RDP, i.e. $\left(\alpha, \alpha\frac{\epsilon^2_{\text{GM}}}{4\ln(1.25/\delta_{\text{GM}})}\right)$-RDP for every $\alpha > 1$ (see Proposition 5).

The Exponential Mechanism (Definition 6) is $\frac{\epsilon^2_{\text{EM}}}{8}$- concentrated DP [51, Corollary 7]. Proposition 2 shows that ρ-CDP implies RDP by definition — because every ρ-CDP mechanism satisfies RDP with $\epsilon(\alpha) = \rho\alpha\ \forall\alpha > 1$. Then, the Exponential Mechanism satisfies $\left(\alpha, \alpha\frac{\epsilon^2_{\text{EM}}}{8}\right)$-RDP.

TRIP* executes the Exponential Mechanism L times per round, since $\Pi^{\text{DP}}_{\texttt{qRankMPC}}$ runs once in every round of TRIP*. Furthermore, TRIP* executes the Gaussian mechanism once per round. Since the total number of rounds is T and the total RDP cost is additive under composition (see Proposition 3), it holds that

$$\text{TRIP* satisfies } \left(\alpha,\ T\alpha\frac{\epsilon^2_{\text{GM}}}{4\ln(1.25/\delta_{\text{GM}})} + TL\alpha\frac{\epsilon^2_{\text{EM}}}{8}\right)\text{-RDP.}$$

Finally, converting from RDP to (ϵ, δ)-DP via Proposition 4 yields that for any $\alpha > 1$ and $0 < \delta < 1$, TRIP* satisfies $(\epsilon(\alpha), \delta)$-differential privacy for

$$\epsilon(\alpha) = T\alpha\frac{\epsilon^2_{\text{GM}}}{4\ln(1.25/\delta_{\text{GM}})} + TL\alpha\frac{\epsilon^2_{\text{EM}}}{8} + \frac{\ln(1/\delta)}{\alpha-1} \tag{6}$$

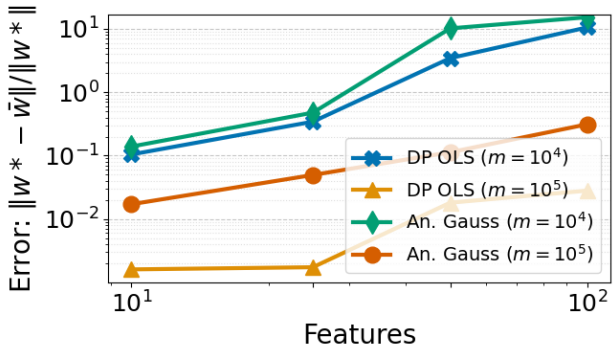
We achieve the tightest achievable (ϵ, δ)-DP guarantee in our setting by minimizing $\epsilon(\alpha)$ for $\alpha > 1$. We fix the parameters T, L and determine the optimal value of α numerically (using, based on the ℓ_2-sensitivity Δ of the OLS estimate $\bar{\mathbf{w}}$, the noise of the Gaussian mechanism σ, the desired overall privacy guarantee ϵ, and the failure probability $\delta_{\text{final}} = \delta + \delta'$. In practice, δ_{final} depends on the dataset size and is chosen such that $\delta_{\text{final}} < 1/m$.

Privacy Budget Selection. The choice of ϵ depends on the privacy-utility trade-off: smaller values strengthen privacy but reduce utility. The community generally recommends $\epsilon < 10$, ideally $\epsilon \leq 5$, as meaningful protection, consistent with major real-world deployments [5,34,37]. For δ, values between 10^{-5} and 10^{-9} are standard. These thresholds also align with practices from major real-world deployments (e.g., Facebook [34], Apple [5], Google [57]), where ϵ in the low single digits is considered strong privacy. Throughout our experiments, we fix $\delta < 1/m$, vary ϵ, and decrease it until TRIP* fails to converge in high-dimensional settings.

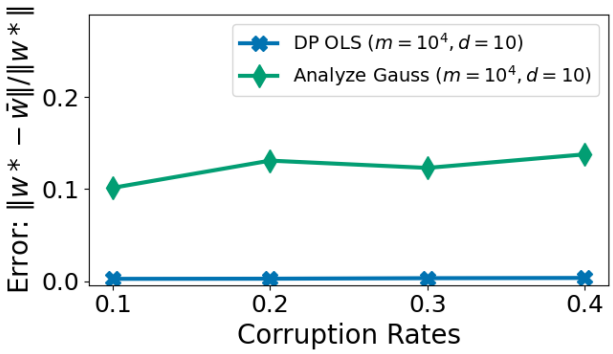
6 Evaluation

We evaluate our protocol in terms of accuracy and efficiency on synthetic and real-world data ([14,18]). We measure the accuracy of our algorithm as the average over 10 runs of the ℓ_2 distance between the ground truth $\mathbf{w}^*$ and the output model $\bar{\mathbf{w}}$. In Section 6.1, we evaluate the accuracy of two different DP techniques for our algorithm on simulated data. We also assess TRIP*'s accuracy in real-world datasets. In Section 6.2, we benchmark the efficiency of our protocol and compare it with the baseline.

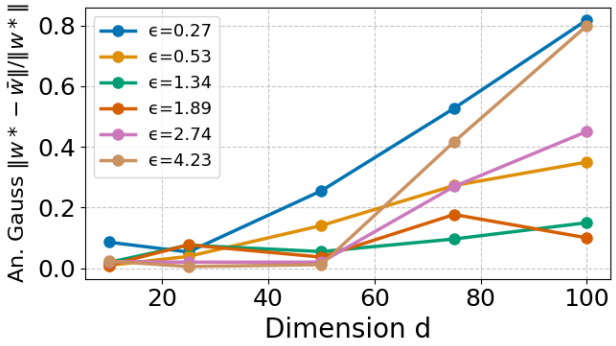
6.1 Accuracy Evaluation



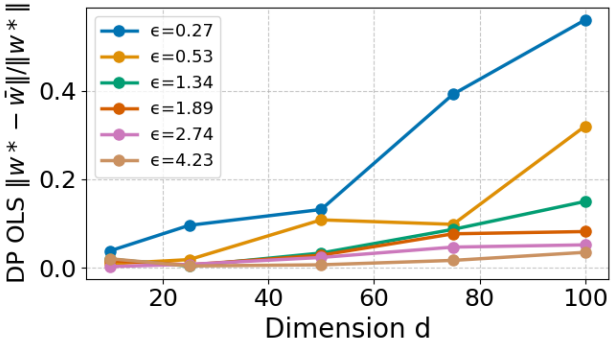
(a) Varying features and samples, $\epsilon \approx 2.7$



(b) Varying corruptions β, $\epsilon \approx 2.7$



(c) Privacy budgets using [27]



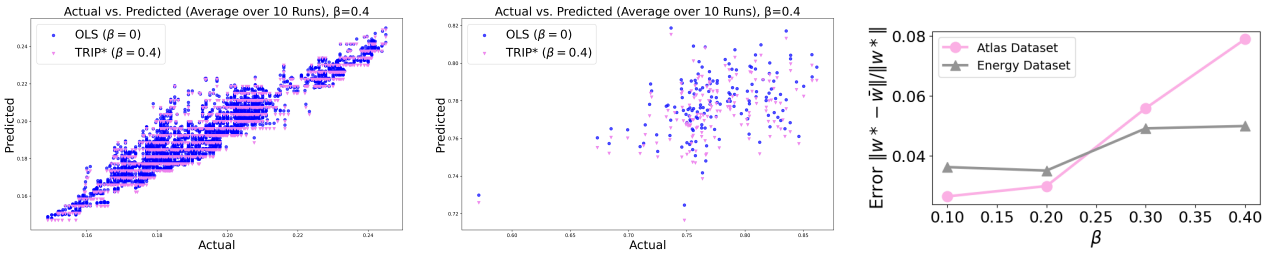
(d) Privacy budgets using proposition 1 and lemma 2

Fig. 9: [Synthetic Data] Error between TRIP*'s model and ground truth.

Synthetic Data To evaluate our robust linear regression algorithm, we analyze the convergence of the model on a synthetic Gaussian dataset. The dataset consists of observations $(\mathbf{x}_i, y_i)$ where each $\mathbf{x}_i \in \mathbb{R}^d$ has independent standard Gaussian entries. The labels are generated as: $y_i = \mathbf{x}_i^\top \mathbf{w}^* + z_i$ where $\mathbf{w}^*$ is sampled from a standard Gaussian, and z_i, representing observation noise, is drawn from a Gaussian distribution with standard deviation $\sigma = 0.2$.

Adversarial Corruption Simulation Corruptions are introduced using an adversarial model, targeting attacks that push the model away from the ground truth towards an adversarially chosen one in a way that is not easily detectable. First, an adversarial $\mathbf{w}_\mathcal{A}$ is chosen, and then, for corrupted points, the response is generated as $y_i^\mathcal{A} = \mathbf{x}_i^\top \mathbf{w}_\mathcal{A} + \phi|\mathbf{x}_i^\top \mathbf{w}_\mathcal{A} - y_i|$, overwriting the true label. Here, ϕ is a scaling factor (e.g. $\phi = 1.5$) that pushes the model towards $\mathbf{w}_\mathcal{A}$. We consider a powerful adversary able to choose the corruption locations, after viewing $\mathbf{x}, \mathbf{y}$ and $\mathbf{w}$, as described in Section F. To empirically verify the convergence rate and fix a constant number of rounds in our protocol, we devise several strategies for choosing $\mathbf{w}_\mathcal{A}$ concluding that 5 rounds of the Robust Model Update Phase suffice. For a more detailed analysis of the robustness of TRIP* on synthetic data with varying numbers of features and corruptions under fine-grained attack strategies we refer the reader to Sections E and F.

Sensitivity to Dimension and Corruption We generate synthetic datasets of $m = 10^5$ and $m = 10^4$ from a standard Gaussian, varying the feature dimension d and corruption fraction β to assess TRIP*'s robustness with fixed-point arithmetic and DP noise. Figures 9a and 9b show that TRIP* tolerates large corruption



(a) Energy Dataset: TRIP* vs. OLS predictions ($\beta = 0.3$).

(b) Opportunity Atlas: TRIP* vs. OLS predictions ($\beta = 0.3$).

(c) Error between TRIP*'s approximation and OLS approximation.

Fig. 10: Real-world evaluation of TRIP* on corrupted datasets ($\beta = 0.3$) against a clean OLS baseline.

fractions and is not sensitive to d for $m = 10^5$. Our conclusions extend to larger datasets, since TORRENT's convergence rate scales as $\sqrt{m}$ [8, Theorem 10].

Differentially Private Model TRIP* runs over multiple rounds, with the Gaussian and Exponential mechanisms consuming per-round budgets ϵ_{GM} and ϵ_{EM}, and total budget ϵ computed via Equation (5). With $\epsilon_{\text{GM}} = 1$, $\epsilon_{\text{EM}} = 0.0625$, and 5 and $5L = 80$ rounds respectively, we achieve $\epsilon \approx 2.7$ (see Figures 9a and 9b). For $m = 10^4$ accuracy holds up to $d = 25$, while $m = 10^5$ scales to $d = 100$. For the larger sample size, high accuracy is achievable even under a tighter budget of $\epsilon \approx 1.34$ (see Figures 9c and 9d).

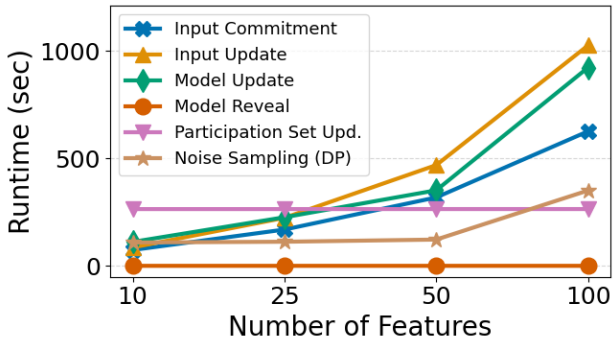
Real-World Datasets We evaluate TRIP* on two real-world datasets [14, 18], using an 80/20 train/test split and the same DP parameters as in Section 6.1. In both cases, we define $\mathbf{w}^*$ as the OLS model trained on the clean training sets. Before applying TRIP*, we validate that linear regression is a well-fitted model for both datasets, as shown in Figures 10a and 10b. Figure 10c shows that the relative model error $\|\bar{\mathbf{w}} - \mathbf{w}^*\|/\|\mathbf{w}^*\|$ remains below 0.1 when corrupting up to 40% of target values with adversarial responses drawn uniformly from a bounded range.

Energy Dataset. We use the Appliances Energy Consumption Dataset [14] to predict room temperature from a single input feature (temperature of a different room), with $m_{\text{train}} = 15{,}788$ and $m_{\text{test}} = 3{,}947$ samples. We normalize all values to $[0, 1]$, giving sensitivities of 1 for both $\boldsymbol{A}$ and $\mathbf{b}$, and corrupt up to 40% of target values with adversarial responses drawn uniformly from $[0.5, 1]$. TRIP* closely matches the OLS model on clean data across all tested corruption levels $\beta \in [0.1, 0.4]$.

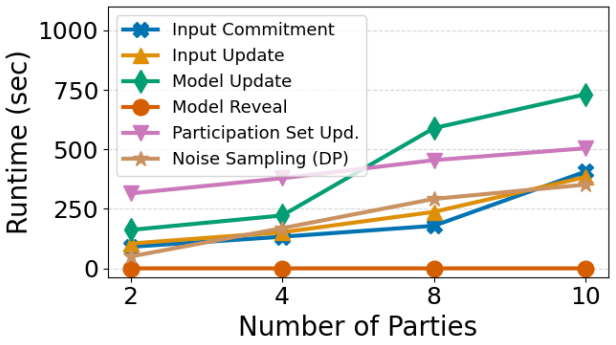
Opportunity Atlas. We use data from [18] to predict the fraction of people with positive W-2 earnings in 2015 from their W-2 earnings fraction at age 26, across Census tracts, with $m_{\text{train}} = 592$ and $m_{\text{test}} = 149$ samples. Since all variables are fractions in $[0, 1]$, no normalization is needed and sensitivities of $\boldsymbol{A}$ and $\mathbf{b}$ are both 1. Corruptions are modeled as in the Energy Dataset, and TRIP* again achieves accuracy comparable to OLS on the clean data.

6.2 Efficiency Benchmarks for TRIP*

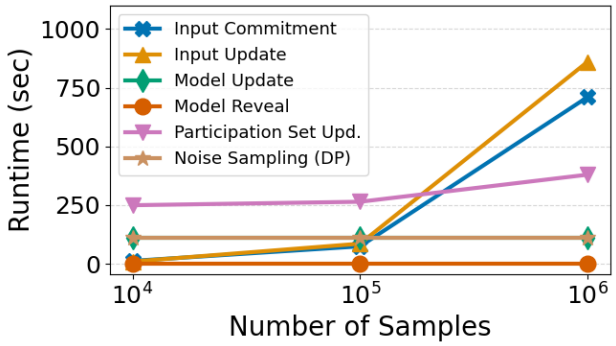
To evaluate scalability, we vary the number of samples (n), features (d), number of parties (N). To ensure compatibility with cryptographic primitives, which rely on group and field operations, we represent individuals' datasets using a fixed-point integer format. We assume that the necessary setup has been completed before the protocol execution starts. Specifically, the parties have established a secure communication channel (e.g. using SSL) and generated the required encryption keys. We implement our protocol using the MP-SPDZ library [38] and the EMP-TOOL library [56]. The computational security parameter κ is fixed at 128 and $\rho = 40$ bits are used for statistical security in both protocols. Finally, we use a 61-bit field in ZK, a 128-bit prime modulo p and 16 bits for the fixed-point precision in MPC.



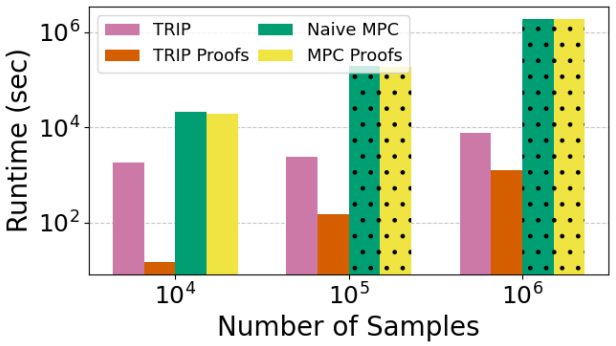
(a) Per-phase breakdown with varying features ($m = 10^5, N = 3$).



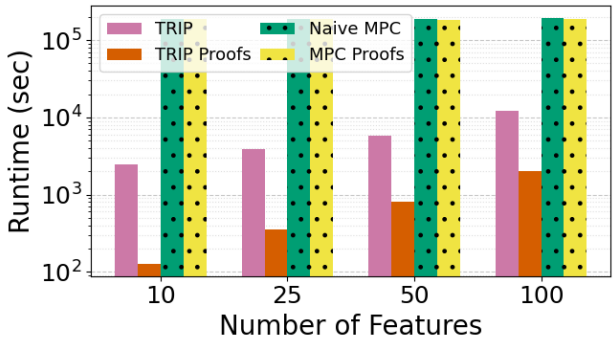
(b) Per-phase breakdown with varying parties ($m = 10^5, d = 10$).



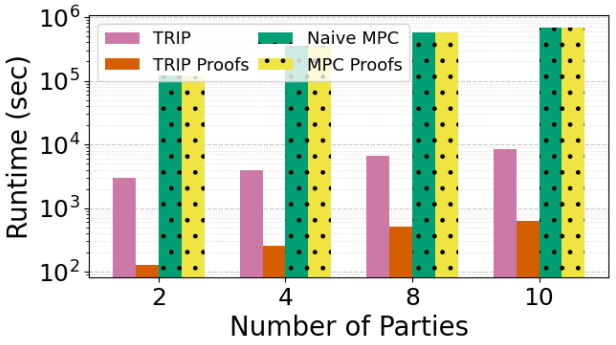
(c) Per-phase breakdown with varying samples ($d = 10, N = 3$).



(d) Runtime vs. samples ($d = 10, N = 3$).



(e) Runtime vs. features ($m = 10^5, N = 3$).



(f) Runtime vs. parties ($m = 10^5, d = 10$).

Fig. 11: Runtime benchmarks for TRIP*. Per-phase runtime of is for 1 round. Comparison is done against MPC-only baseline; dotted bars denote simulated runtimes where direct computation is infeasible.

Experimental Setting and Observations We evaluate TRIP*'s scalability under varying samples, features, and parties, running the Robust Model Update Phase for 5 rounds. Timings include preprocessing, and experiments were conducted on AWS c5.12xlarge instances across Oregon and Ohio to simulate a WAN. Results are summarized in Figure 11.

As shown in Figure 11c, for varying m, the Input Update phase is the main bottleneck, as each party must share inputs and prove correctness of their summaries. Input Commitment also scales with m but runs only once. The Participation Set Update is dominated by the quantile computation –executed for a constant 16 rounds, independent of m– while the commitment on residuals (of size n each) and the range proofs add to the runtime, explaining the modest increase observed.

For varying d (see Figure 11a), the Model Compute Phase dominates with $\mathcal{O}(d^2)$ cost from five rounds of ADMM. Input Commitment scales with $m \times d$ and requires d^2 matrix multiplication proofs in ZK, explaining the quadratic scaling. The Noise Sampling Phase generates random bits in batch, causing the observed peak at $d = 100$.

For varying N (see Figure 11b), overhead again comes primarily from the Model Compute Phase over secret-shared data. The Participation Set Update runs for a fixed number of 16 rounds, during which parties perform only $\mathcal{O}(1)$ comparisons and exponentiations; the remaining computation is carried out locally on individual data. This explains the modest increase in overhead as the number of parties grows. Note that the per-phase times correspond to a single round of TRIP*, while the comparison with the MPC-only baseline reflects the full protocol execution.

Comparison with MPC baseline We compare TRIP* against an MPC-only baseline where all correctness proofs are performed in MPC instead of ZK. We apply the same optimizations to both, including our optimized threshold computation and use of DP. We note that an implementation where all intermediate models and thresholds are secret-shared is prohibitively inefficient, and we estimate a 600× overhead over TRIP* for 10^6 samples. Both protocols use the Overdrive [39] (low-gear) protocol in MP-SPDZ, with timings including preprocessing. TRIP* achieves speedups of 10×, 80×, and 250× for 10^4, 10^5, and 10^6 samples respectively, with gains increasing with dataset size as we reduce the linear dependence on m. For more than three parties at $m = 10^5$, TRIP* is 40–80× faster. For $d = 10$, we obtain an 80× speedup, which decreases as d grows due to the quadratic MPC scaling in d. Results appear in Figures 11d to 11f.

7 Related Work

Mitigating malformed inputs in secure computation is an active research area. [20] uses semi-honest servers to test input well-formedness, with follow-up works improving efficiency [1] and extending to federated learning [50]. We consider a stronger model where all-but-one parties are fully malicious; in this setting, [15] implements statistical tests to identify biased datasets and [60] trains a linear model. Many works use secure computation for privacy-preserving machine learning (e.g., [16,40]), covering settings from semi-honest to malicious security, two- and multi-party computation, and tasks ranging from linear models to neural network inference. [16] avoids poisoning via ensembling, but assumes all parties have representative data an assumption we do not make. A related line of work considers DP for model training [4, 49]. [4] evaluates DP algorithms for linear regression; [42] integrates DP with MPC but targets a different objective and does not provide robustness guarantees; [49] studies linear regression under DP with outliers but without cryptographic privacy.

8 Limitations and Conclusions

We construct TRIP* to address the issue of malicious inputs in linear regression tasks. We mitigate information leakage by obscuring the intermediate models and the quantiles by incorporating multi-party computation (MPC) security with differential privacy (DP). Achieving comparable efficiency while avoiding DP requires an efficient multi-party sorting protocol with communication sublinear to the input size, as well as a scoring function that can be computed in a secret-shared manner independently of the input size. To our knowledge, no such methods currently exist. We emphasize that the choice and interpretation of the DP budget remain active topics of research today. The parties must therefore agree between themselves on a suitable privacy budget based on the currently known attack landscape. Similarly, users of the protocol must take particular care in

understanding and managing the cryptographic and DP guarantees provided. To conclude, we consider a real threat in practical systems where a powerful adversary can corrupt part of the input. Overall, TRIP* provides a practical step toward secure, robust, and privacy-preserving collaborative learning.

Acknowledgments. This work has been partially supported by project MIS 5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the European Union under the NextGenerationEU Program.

References

1. Addanki, S., Garbe, K., Jaffe, E., Ostrovsky, R., Polychroniadou, A.: Prio+: Privacy preserving aggregate statistics via boolean shares. In: SCN 2022 (2022)
2. Aggarwal, G., Mishra, N., Pinkas, B.: Secure computation of the k th-ranked element. In: EUROCRYPT (2004)
3. Aggarwal, G., Mishra, N., Pinkas, B.: Secure Computation of the Median (and Other Elements of Specified Ranks). Journal of Cryptology (2010)
4. Alabi, D., McMillan, A., Sarathy, J., Smith, A., Vadhan, S.: Differentially private simple linear regression. PoPETS (2022)
5. Apple Inc.: Differential privacy (2022)
6. Bater, J., He, X., Ehrich, W., Machanavajjhala, A., Rogers, J.: Shrinkwrap: efficient sql query processing in differentially private data federations. Proc. VLDB Endow. (2018)
7. BBC: ChatGPT banned in Italy over privacy concerns (2023)
8. Bhatia, K., Jain, P., Kar, P.: Robust regression via hard thresholding. In: NeurIPS (2015)
9. Böhler, J., Kerschbaum, F.: Secure multi-party computation of differentially private median. USENIX Security (2020)
10. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers (2011)
11. Bun, M., Steinke, T.: Concentrated differential privacy: Simplifications, extensions, and lower bounds. In: TCC (2016)
12. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC (2002)
13. Canonne, C., Kamath, G., Steinke, T.: Discrete Gaussian for differential privacy. Journal of Privacy and Confidentiality (2022)
14. Chandra, M.G.: Appliances energy consumption dataset (2022)
15. Chang, I., Sotiraki, K., Chen, W., Kantarcioglu, M., Popa, R.: {HOLMES}: Efficient Distribution Testing for Secure Collaborative Learning. In: USENIX Security (2023)
16. Chaudhari, H., Jagielski, M., Oprea, A.: Safenet: The unreasonable effectiveness of ensembles in private collaborative learning. In: IEEE Conference on Secure and Trustworthy Machine Learning (2023)
17. Chen, H., Kim, M., Razenshteyn, I., Rotaru, D., Song, Y., Wagh, S.: Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning. In: ASIACRYPT (2020)
18. Chetty, R., Friedman, J.N., Hendren, N., Jones, M.R., Porter, S.R.: The opportunity atlas: Mapping the childhood roots of social mobility. Tech. rep., National Bureau of Economic Research (2018)
19. Cline, A.K.: Rate of convergence of lawson's algorithm. Mathematics of Computation (1972)
20. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: USENIX NSDI (2017)
21. Damgård, I., Nielsen, J.B.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: Boneh, D. (ed.) CRYPTO (2003)
22. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption. CRYPTO (2012)
23. David, B., Dowsley, R., Katti, R., Nascimento, A.C.A.: Efficient Unconditionally Secure Comparison and Privacy Preserving Machine Learning Classification Protocols. In: Provable Security (2015)
24. Diakonikolas, I., Kamath, G., Kane, D., Li, J., Steinhardt, J., Stewart, A.: Sever: A robust meta-algorithm for stochastic optimization. In: ICML (2019)
25. Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., Naor, M.: Our data, ourselves: Privacy via distributed noise generation. In: EUROCRYPT (2006)
26. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. (2014)
27. Dwork, C., Talwar, K., Thakurta, A., Zhang, L.: Analyze gauss: optimal bounds for privacy-preserving principal component analysis. STOC (2014)
28. Evans, D., Kolesnikov, V., Rosulek, M.: Defining multi-party computation. In: A Pragmatic Introduction to Secure Multi-Party Computation (2018)

29. FTC: FTC takes action against company formerly known as weight watchers for illegally collecting kids sensitive health data (2022)
30. Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., Evans, D.: Privacy-preserving distributed linear regression on high-dimensional data. PoPETS (2017)
31. Goldreich, O.: Foundations of Cryptography: Volume 1: Basic Tools. Cambridge University Press
32. Hamada, K., Ikarashi, D., Chida, K., Takahashi, K.: Oblivious radix sort: An efficient sorting algorithm for practical secure multi-party computation. ePrint (2014)
33. Hamada, K., Kikuchi, R., Ikarashi, D., Chida, K., Takahashi, K.: Practically efficient multi-party sorting protocols from comparison sort algorithms. In: ICISC (2012)
34. Herdadelen, A., Dow, A., State, B., Mohassel, P., Pompe, A.: Protecting privacy in Facebook mobility data during the COVID-19 response (2020)
35. Huber, P.J.: Robust estimation of a location parameter. In: Breakthroughs in statistics: Methodology and distribution (1992)
36. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: USENIX Security (2018)
37. Karmitsa, N., Airola, A., Pahikkala, T., Pitkämäki, T.: A comprehensive guide to differential privacy: From theory to user expectations (2025)
38. Keller, M.: MP-SPDZ: A Versatile Framework for Multi-Party Computation. In: ACM SIGSAC CCS (2020)
39. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ Great Again. In: EUROCRYPT (2018)
40. Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., van der Maaten, L.: Crypten: secure multi-party computation meets machine learning. In: NeurIPS (2021)
41. Lawson, C.L.: Contribution to the theory of linear least maximum approximation. Ph. D. dissertation. Univ. Calif. (1961)
42. Liu, X., Jain, P., Kong, W., Oh, S., Suggala, A.: Label robust and differentially private linear regression: Computational and statistical efficiency. In: NeurIPS (2023)
43. Makri, E., Rotaru, D., Smart, N.P., Vercauteren, F.: Epic: efficient private image classification (or: Learning from the masters). In: RSA Conference (2019)
44. Mironov, I.: Rényi differential privacy. In: IEEE CSF (2017)
45. Mohassel, P., Rindal, P.: ABY ^{3}: A Mixed Protocol Framework for Machine Learning. In: ACM SIGSAC CCS (2018)
46. Mukhoty, B., Dey, D., Kar, P.: Corruption-Tolerant Algorithms for Generalized Linear Models. AAAI (2023)
47. Mukhoty, B., Gopakumar, G., Jain, P., Kar, P.: Globally-convergent iteratively reweighted least squares for robust regression problems. In: AISTATS (2019)
48. Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., Taft, N.: Privacy-preserving ridge regression on hundreds of millions of records. In: IEEE S&P (2013)
49. Pentyala, S., Railsback, D., Maia, R., Dowsley, R., Melanson, D., Nascimento, A., Cock, M.D.: Training differentially private models with secure multiparty computation (2025)
50. Rathee, M., Shen, C., Wagh, S., Popa, R.A.: ELSA: secure aggregation for federated learning with malicious actors. In: IEEE S&P (2023)
51. Rogers, R., Steinke, T.: A better privacy analysis of the exponential mechanism. DifferentialPrivacy.org (2021)
52. Rotaru, D., Smart, N.P., Tanguy, T., Vercauteren, F., Wood, T.: Actively secure setup for spdz. Journal of Cryptology (2022)
53. Tukey, J.W.: A survey of sampling from contaminated distributions. Contributions to probability and statistics (1960)
54. Vershynin, R.: Introduction to the non-asymptotic analysis of random matrices (2011)
55. Wagh, S., He, X., Machanavajjhala, A., Mittal, P.: DP-cryptography: marrying differential privacy and cryptography in emerging applications. Communications of the ACM (2021)
56. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: Efficient MultiParty computation toolkit (2016)
57. Xu, Z., Zhang, Y., Andrew, G., Choquette, C., Kairouz, P., Mcmahan, B., Rosenstock, J., Zhang, Y.: Federated learning of gboard language models with differential privacy. In: Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track) (2023)
58. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In: ACM SIGSAC CCS (2021)
59. Zheng, W., Deng, R., Chen, W., Popa, R.A., Panda, A., Stoica, I.: Cerebro: A platform for Multi-Party cryptographic collaborative learning. In: USENIX Security (2021)
60. Zheng, W., Popa, R.A., Gonzalez, J.E., Stoica, I.: Helen: Maliciously secure coopetitive learning for linear models. In: IEEE S&P (2019)

A Missing Definitions

A.1 Definition of MPC

Definition 1 (MPC (See [31])). *We define* multiparty secure computation (MPC) *(with abort) using the ideal/real world paradigm in the UC framework.*

Real world. Let Π be a N-party protocol among $(\mathcal{P}_1, \ldots, \mathcal{P}_N)$ computing a N-party function $f : (\{0,1\}^*)^N \to (\{0,1\}^*)^N$, let $\mathcal{C} \subseteq [N]$ denote the set of indices of the corrupted parties, and let $\mathcal{Z}$ be an interactive Turing machine representing the environment. At setup, $\mathcal{Z}$ provides $(1^\lambda, \mathbf{x}_i)$ to each party $\mathcal{P}_i$ and $(\mathcal{C}, \{\mathbf{x}_i\}_{i\in\mathcal{C}}, \text{aux})$ to the adversary $\mathcal{A}$, where aux *denotes some auxiliary input. The real-world execution on input $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and security parameter λ with respect to $(\mathcal{Z}, \mathcal{A}, \mathcal{C})$, denoted by $\mathsf{real}_{\Pi,\mathcal{C},\mathcal{A}(\text{aux})}(\mathbf{x}, \lambda)$, is the output of $\mathcal{Z}$ resulting from the protocol interaction.*

Ideal world. Let $f : (\{0,1\}^*)^N \to (\{0,1\}^*)^N$ be a N-party function and let $\mathcal{Z}$ be an interactive Turing machine, representing the environment, that at setup provides $(1^\lambda, \mathbf{x}_i)$ to each party $\mathcal{P}_i$ and $(\mathcal{C}, \{\mathbf{x}_i\}_{i\in\mathcal{C}}, \text{aux})$ to the simulator S, where $\mathcal{C} \subseteq [N]$ represents the set of corrupted parties and aux *is some auxiliary input. Similarly to the real-world execution, the ideal-world execution on input $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and security parameter λ with respect to $(\mathcal{Z}, \mathsf{S}, \mathcal{C})$, denoted by $\mathsf{ideal}_{f,\mathcal{C},\mathsf{S}(\text{aux})}(\mathbf{x}, \lambda)$, is the output of $\mathcal{Z}$ from the following process:*

- Trusted party receives inputs: *Each honest party $\mathcal{P}_i$ sends to the trusted party its input $\mathbf{x}_i$. The simulator S may send to the trusted party arbitrary inputs corresponding to the corrupted parties $\{\mathbf{x}_i\}_{i\in\mathcal{C}}$.*
- Trusted party computes outputs: *The trusted party computes $(y_1, \ldots, y_N) \stackrel{def}{=} f(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and sends $\{y_i\}_{i\in\mathcal{C}}$ to the simulator S.*
- Simulator decides to abort or not: *The simulator S responds with* abort *or $\perp$.*
- Trusted party answers to honest parties: *If the trusted party received $\perp$, then it sends $\perp$ to each honest party $\mathcal{P}_i$. Otherwise, it sends y_i to each party $\mathcal{P}_i$.*
- Environment computes output: *The environment $\mathcal{Z}$ receives the outputs of the honest parties and an arbitrary function of S's view of the protocol. Finally, $\mathcal{Z}$ outputs a bit.*

An MPC protocol is secure if for every probabilistic polynomial-time adversary $\mathcal{A}$ there exists a simulator S such that the distributions

$$\{\mathsf{real}_{\Pi,\mathcal{C},\mathcal{A}(\text{aux})}(\mathbf{x}, \lambda)\}_{\mathbf{x}\in(\{0,1\}^*)^N, \lambda\in\mathbb{N}}$$

and

$$\{\mathsf{ideal}_{f,\mathcal{C},\mathsf{S}(\text{aux})}(\mathbf{x}, \lambda)\}_{\mathbf{x}\in(\{0,1\}^*)^N, \lambda\in\mathbb{N}}$$

are (computationally) indistinguishable.

A.2 Zero-knowledge proofs

Definition 2 (ZK (See [31])). *Let Π be an interactive protocol, between a prover P and a verifier V. Let a language L in the complexity class NP ($L \in \mathsf{NP}$) characterized by a polynomial-time relation R_L such that $L = \{x \mid \exists\, w : (x, w) \in R_L\}$. Π is called an interactive (computationally) zero knowledge proof for the language L if it satisfies the following properties:*

1. Completeness. *For all pairs $(x, w) \in R_L$, if P, V have inputs (x, w), x respectively and they run Π, then V outputs 1 (V accepts the proof).*
2. Soundness. *For all $x \notin L$, for every positive polynomial $p(\cdot)$, for all sufficiently large k, and for all polynomial time algorithms $\mathcal{A}$, on input x, if $\mathcal{A}$, V have inputs $(x, 1^k)$, x respectively and they run Π, then V outputs 0 except with negligible probability $1/p(k)$ (V rejects the proof).*
3. Zero Knowledge. *For every probabilistic polynomial-time interactive machine V^*, there is a probabilistic polynomial-time algorithm* `Sim` *acting as a simulator for the interaction of P and V^*, such that the output of the interactive machine V^* after it interacts with P (on common input x) and the output of the simulator* `Sim` *(on input x) are computationally indistinguishable, $\forall\, x \in L$ (whatever can be efficiently computed after the interaction of V^* with P on common input $x \in L$ can also be efficiently computed from x itself).*

A.3 Differential Privacy

Definition 3 (Global l_2 Sensitivity (See [26])). *Let $f : \mathcal{X}^n \to \mathcal{R}^k$, the l_2-sensitivity of f is $\Delta f = \max_{X,X'} \|f(X) - f(X')\|_2$, where X and X' are neighbouring inputs.*

Definition 4 (Gaussian Mechanism (See [26])). *Let $f : \mathcal{X}^n \to \mathcal{R}^k$. The Gaussian mechanism is defined as $\mathcal{N}(X) = f(X) + (Y_1, \cdots, Y_k)$, where the Y_i are independent $\mathcal{N}\left(0, 2\ln(\frac{1.25}{\delta})\frac{\Delta f^2}{\epsilon^2}\right)$ random variables.*

Definition 5 (Global Sensitivity of Utility Function (See [26])). *Let a utility function $\mathrm{u}(X, r)$ that takes as input X, and a possible output $r \in \mathcal{R}$. The global sensitivity Δu is defined as:*

$$\Delta u = \max_{r \in \mathcal{R}} \max_{X,X'} |\mathrm{u}(X, r) - \mathrm{u}(X', r)|,$$

where X and X' are neighbouring inputs.

Definition 6 (Exponential Mechanism (See [26])). *Assume a fixed input $X \in \mathcal{X}^n$. The exponential mechanism $\mathcal{E}(X, \mathcal{R})$ selects and outputs an element $r \in \mathcal{R}$ with probability proportional to the quantity $\exp\left(\frac{\epsilon \cdot \mathrm{u}(X,r)}{2\Delta u}\right)$*

Definition 7 (Rényi Divergence of order α (See [44])). *For two probability distributions P and Q defined over $\mathbb{R}$, the Rényi divergence of order $\alpha > 1$ is:*

$$D_\alpha(P\|Q) \stackrel{def}{=} \frac{1}{\alpha-1} \ln E_{x\sim Q}\left(\frac{P(x)}{Q(x)}\right)^\alpha$$

where $P(x), Q(x)$ are the densities of P, Q, respectively, at x.

Definition 8 ((α, ϵ)-Rényi Differential Privacy (See [44])). *A randomized mechanism $f : \mathcal{D} \to \mathbb{R}$ is said to have ϵ-Rényi differential privacy of order α, or (α, ϵ)-RDP, if for any neighboring $D, D' \in \mathcal{D}$ it holds that:*

$$D_\alpha(f(D)\|f(D')) \le \epsilon$$

Definition 9 (ρ-Concentrated Differential Privacy (ρ-CDP) (see [11])). *A randomised mechanism $f : \mathcal{D} \to \mathbb{R}$ is said to be ρ-zero-concentrated differentially private (ρ-zCDP) if, for any neighboring $D, D' \in \mathcal{D}$ and all $\alpha \in (1, \infty)$, it holds that:*

$$D_\alpha(f(D)\|f(D')) \le \rho\alpha$$

where $D_\alpha(f(D)\|f(D'))$is the Rényi divergence between the distribution of $f(D)$ and the distribution of $f(D')$.

Proposition 2 (ρ-CDP implies Rényi DP). *If a randomized mechanism f satisfies ρ-CDP, then for all $\alpha > 1$, f satisfies $(\alpha, \varepsilon(\alpha))$-RDP with $\epsilon(\alpha) = \alpha\rho$.*

Proof. Fix $\alpha > 1$ and neighboring datasets D, D'. By ρ-CDP, $D_\alpha(f(D)\|f(D')) < \rho\alpha$. Setting $\epsilon(\alpha) \stackrel{\text{def}}{=} \rho\alpha$, this is exactly the definition of $(\alpha, \epsilon(\alpha))$-RDP. Since α was arbitrary, the result holds for all $\alpha > 1$.

Proposition 3 (Composition under Rényi Differential Privacy). *(See [44, Proposition 1]) Let $f : \mathcal{D} \to \mathbb{R}_1$ be (α, ϵ_1)-RDP and $g : \mathbb{R}_1 \times \mathcal{D} \to \mathbb{R}_2$ be (α, ϵ_2)-RDP, then the mechanism defined as (X, Y), where $X \sim f(D)$ and $Y \sim g(X, D)$, satisfies $(\alpha, \epsilon_1 + \epsilon_2)$-RDP*

Proposition 4 (Rényi Differential Privacy to (ϵ, δ) Differential Privacy). *(See [44, Proposition 3]) Let $f : \mathcal{D} \to \mathbb{R}$ be (α, ϵ)-RDP. Then f satisfies $(\epsilon + \frac{\ln(1/\delta)}{\alpha-1}, \delta)$-differential privacy for any $0 < \delta < 1$.*

Proposition 5 (Rényi Differential Privacy guarantee of the Gaussian Mechanism). *(See [11, Proposition 1.6] and Proposition 2) Let $f : \mathcal{X}^n \to \mathbb{R}^k$ with sensitivity Δ. The Gaussian mechanism $\mathcal{N}(X)$ applied on f satisfies $(\alpha, \alpha\Delta^2/(2\sigma^2))$-RDP.*

B Missing Proofs

B.1 Proof of ℓ_2 sensitivity of OLS

Lemma 1. *[Sensitivity of $\bar{\mathbf{w}}$] Let $\boldsymbol{X}$ be an $m \times d$ matrix whose rows $\mathbf{x}_k$ are independent random vectors in $\mathbb{R}^d$ drawn from a distribution $\mathcal{D}$ with the common second moment matrix $\boldsymbol{\Sigma} \stackrel{def}{=} E_{\mathbf{x}\sim\mathcal{D}}[\mathbf{x}\otimes\mathbf{x}]$. Let $\mathbf{y} \in \mathbb{R}^m$. Assume there exist constants B_x, B_y such that $\|\mathbf{x}_k\|_2 \le B_x$ and $|y_k| \le B_y$ almost surely (i.e., with probability 1) for all $k \in [m]$. Then, for every $t \ge 0$, with high probability over the randomness of $\boldsymbol{X}$, the ℓ_2-sensitivity of the OLS estimator $\bar{\mathbf{w}}$ satisfies:*

$$\Delta\bar{\mathbf{w}} \le m \frac{2B_x^3 B_y}{\left(\sqrt{m\,\lambda_{\min}(\boldsymbol{\Sigma})}-tB_x\right)^2\left(\left(\sqrt{m\,\lambda_{\min}(\boldsymbol{\Sigma})}-tB_x\right)^2-2B_x^2\right)} + \frac{2B_xB_y}{\left(\sqrt{m\,\lambda_{\min}(\boldsymbol{\Sigma})}-tB_x\right)^2-2B_x^2} \tag{7}$$

where $\boldsymbol{A} \stackrel{def}{=} \boldsymbol{X}^\top\boldsymbol{X}$, $\mathbf{b} \stackrel{def}{=} \boldsymbol{X}^\top\mathbf{y}$, $\boldsymbol{A}' \stackrel{def}{=} \boldsymbol{X}'^\top\boldsymbol{X}'$, $\mathbf{b}' \stackrel{def}{=} \boldsymbol{X}'^\top\mathbf{y}'$, and $\boldsymbol{A}', \mathbf{b}'$ are defined by changing a single row of the dataset, i.e., $\mathbf{x}_k \to \mathbf{x}'_k$, $y_k \to y'_k$.

Proof. We use the following standard facts from matrix analysis:

1. $\boldsymbol{A}'^{-1} - \boldsymbol{A}^{-1} = -\boldsymbol{A}^{-1}(\boldsymbol{A}' - \boldsymbol{A})\boldsymbol{A}'^{-1}$.
2. $\|\boldsymbol{A}\boldsymbol{B}\|_2 \le \|\boldsymbol{A}\|_2\|\boldsymbol{B}\|_2$.
3. For a symmetric positive definite matrix, singular values equal eigenvalues.
4. $\boldsymbol{A}' = \boldsymbol{A} + (\mathbf{x}'_k\mathbf{x}'^\top_k - \mathbf{x}_k\mathbf{x}_k^\top)$ is a rank-2 perturbation (a difference of two rank-1 matrices) with $\|\boldsymbol{A}' - \boldsymbol{A}\|_2 \le 2B_x^2$. By Weyl's inequality, $\lambda_{\min}(\boldsymbol{A}') \ge \lambda_{\min}(\boldsymbol{A}) - 2B_x^2$.

Step 1: We obtain the inequality as:

$$\begin{aligned}\left\|\boldsymbol{A}'^{-1}\mathbf{b}' - \boldsymbol{A}^{-1}\mathbf{b}\right\|_2 &= \left\|(\boldsymbol{A}'^{-1} - \boldsymbol{A}^{-1})\boldsymbol{X}^\top\mathbf{y} + \boldsymbol{A}'^{-1}(\mathbf{x}'^\top_k y'_k - \mathbf{x}_k^\top y_k)\right\|_2 \\ &\le \left\|\boldsymbol{A}^{-1}(\boldsymbol{A}' - \boldsymbol{A})\boldsymbol{A}'^{-1}\right\|_2\left\|\boldsymbol{X}^\top\mathbf{y}\right\|_2 + \left\|\boldsymbol{A}'^{-1}\right\|_2\left\|\mathbf{x}'_k y'_k - \mathbf{x}_k y_k\right\|_2 \\ &\le \tfrac{1}{\lambda_{\min}(\boldsymbol{A})}\cdot\tfrac{1}{\lambda_{\min}(\boldsymbol{A}')}\cdot 2B_x^2\cdot mB_xB_y + \tfrac{1}{\lambda_{\min}(\boldsymbol{A}')}\cdot 2B_xB_y.\end{aligned}$$

Writing $\lambda_{\min}(\boldsymbol{X}^\top\boldsymbol{X}) \stackrel{\text{def}}{=} \lambda_{\min}(\boldsymbol{A})$ and applying fact (4), provided $\lambda_{\min}(\boldsymbol{X}^\top\boldsymbol{X}) > 2B_x^2$ (guaranteed by the lower bound derived in Step 2 for sufficiently large m):

$$\left\|\boldsymbol{A}'^{-1}\mathbf{b}' - \boldsymbol{A}^{-1}\mathbf{b}\right\|_2 \le m\,\frac{2B_x^3B_y}{\lambda_{\min}(\boldsymbol{X}^\top\boldsymbol{X})(\lambda_{\min}(\boldsymbol{X}^\top\boldsymbol{X})-2B_x^2)} + \frac{2B_xB_y}{\lambda_{\min}(\boldsymbol{X}^\top\boldsymbol{X})-2B_x^2}. \tag{8}$$

Step 2: We will use the bounded rows assumption and the *whitening* technique to obtain a lower bound on $\lambda_{\min}(\boldsymbol{X}^\top\boldsymbol{X})$.

Define $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{\Sigma}^{-1/2}$, whose rows $\mathbf{z}_k = \boldsymbol{\Sigma}^{-1/2}\mathbf{x}_k$ are isotropic:

$$\mathbb{E}[\mathbf{z}_k\otimes\mathbf{z}_k] = \boldsymbol{\Sigma}^{-1/2}\,\mathbb{E}[\mathbf{x}_k\otimes\mathbf{x}_k]\,\boldsymbol{\Sigma}^{-1/2} = \boldsymbol{I}_d.$$

Their norms satisfy:

$$\|\mathbf{z}_k\|_2 = \|\boldsymbol{\Sigma}^{-1/2}\mathbf{x}_k\|_2 \le \|\boldsymbol{\Sigma}^{-1/2}\|_2\,B_x = \lambda_{\min}(\boldsymbol{\Sigma})^{-1/2}B_x.$$

Applying [54, Theorem 5.41] to $\boldsymbol{Z}$ (isotropic rows, bounded norm) gives, with high probability:

$$\mathrm{s}_{\min}(\boldsymbol{Z}) \ge \sqrt{m} - t\,\lambda_{\min}(\boldsymbol{\Sigma})^{-1/2}B_x.$$

Step 3: Since $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{\Sigma}^{-1/2}$, we have the identity $\boldsymbol{X}^\top\boldsymbol{X} = \boldsymbol{\Sigma}^{1/2}\boldsymbol{Z}^\top\boldsymbol{Z}\,\boldsymbol{\Sigma}^{1/2}$.

For any vector $\mathbf{v} \ne \mathbf{0}$:

$$\begin{aligned}\mathbf{v}^\top(\boldsymbol{X}^\top\boldsymbol{X})\mathbf{v} = \mathbf{v}^\top\boldsymbol{\Sigma}^{1/2}\boldsymbol{Z}^\top\boldsymbol{Z}\,\boldsymbol{\Sigma}^{1/2}\mathbf{v} &\ge \lambda_{\min}(\boldsymbol{Z}^\top\boldsymbol{Z})\|\boldsymbol{\Sigma}^{1/2}\mathbf{v}\|_2^2 \\ &\ge \lambda_{\min}(\boldsymbol{Z}^\top\boldsymbol{Z})\,\lambda_{\min}(\boldsymbol{\Sigma})\,\|\mathbf{v}\|_2^2,\end{aligned}$$

so $\lambda_{\min}(\boldsymbol{X}^\top\boldsymbol{X}) \ge \lambda_{\min}(\boldsymbol{\Sigma})\cdot\lambda_{\min}(\boldsymbol{Z}^\top\boldsymbol{Z})$. Since $\lambda_{\min}(\boldsymbol{Z}^\top\boldsymbol{Z}) = \mathrm{s}_{\min}(\boldsymbol{Z})^2$, combining with Step 2:

$$\lambda_{\min}(\boldsymbol{A}) \ge \lambda_{\min}(\boldsymbol{\Sigma})\left(\sqrt{m} - t\,\lambda_{\min}(\boldsymbol{\Sigma})^{-1/2}B_x\right)^2 = \left(\sqrt{m\,\lambda_{\min}(\boldsymbol{\Sigma})} - tB_x\right)^2. \tag{9}$$

Step 4: Substituting (9) into (8):

$$\Delta\bar{\mathbf{w}} \le m \frac{2B_x^3 B_y}{\left(\sqrt{m\,\lambda_{\min}(\boldsymbol{\Sigma})}-tB_x\right)^2\left(\left(\sqrt{m\,\lambda_{\min}(\boldsymbol{\Sigma})}-tB_x\right)^2-2B_x^2\right)} + \frac{2B_xB_y}{\left(\sqrt{m\,\lambda_{\min}(\boldsymbol{\Sigma})}-tB_x\right)^2-2B_x^2}$$

Lemma 2 (Sensitivity of OLS solution under Isotropy Assumption). *Let $\boldsymbol{X}, \mathbf{y}$ a dataset satisfying all the conditions of Lemma 1. In addition, $\boldsymbol{\Sigma} = \boldsymbol{I}_d$. Then, for any neighboring dataset $(\boldsymbol{X}', \mathbf{y}')$ differing in a single row, for every $t \geq 0$, the following inequality holds w.h.p for the ℓ_2-sensitivity of $\bar{\mathbf{w}}$ derived from the OLS solution:*

$$\Delta\bar{\mathbf{w}} = \max_{\substack{\boldsymbol{X},\boldsymbol{X}' \\ \mathbf{y},\mathbf{y}'}} \left\| \boldsymbol{A}'^{-1}\mathbf{b}' - \boldsymbol{A}^{-1}\mathbf{b} \right\|_2 \leq \; m \, \frac{2B_x^3 B_y}{(\sqrt{m}-tB_x)^2\left((\sqrt{m}-tB_x)^2 - 2B_x^2\right)} + \frac{2B_x B_y}{(\sqrt{m}-tB_x)^2 - 2B_x^2}, \tag{10}$$

where $\boldsymbol{A} \overset{def}{=} \boldsymbol{X}^\top \boldsymbol{X}$, $\mathbf{b} \overset{def}{=} \boldsymbol{X}^\top \mathbf{y}$, $\boldsymbol{A}' \overset{def}{=} \boldsymbol{X}'^\top \boldsymbol{X}'$, $\mathbf{b}' \overset{def}{=} \boldsymbol{X}'^\top \mathbf{y}'$.

Proof. We begin by applying the same technique as in step 1 of Lemma 1, which yields an inequality of the form (8), where the parameters depend on the minimum eigenvalue of $\boldsymbol{X}^\top \boldsymbol{X}$.

Next, we bound $\lambda_{\min}(\boldsymbol{X}^\top \boldsymbol{X})$. Since $\boldsymbol{\Sigma} = \boldsymbol{I}_d$, the rows of $\boldsymbol{X}$ are already isotropic, and [54, Theorem 5.41] applies directly to $\boldsymbol{X}$. This gives, with high probability,

$$\mathrm{s}_{\min}(\boldsymbol{X}) \geq \sqrt{m} - tB_x,$$

which in turn implies

$$\lambda_{\min}(\boldsymbol{X}^\top \boldsymbol{X}) \geq \left(\sqrt{m} - tB_x\right)^2. \tag{11}$$

Substituting (11) into (8):

$$\Delta\bar{\mathbf{w}} \leq m \, \frac{2B_x^3 B_y}{(\sqrt{m}-tB_x)^2\left((\sqrt{m}-tB_x)^2 - 2B_x^2\right)} + \frac{2B_x B_y}{(\sqrt{m}-tB_x)^2 - 2B_x^2},$$

B.2 Proof of Security

$\mathcal{F}_{\text{TRIP}^*}$ - Distributed Robust Regression

Init: Upon receiving $(\text{TRIP}^*, (\boldsymbol{X}_i, \mathbf{y}_i), T, K, \beta, \eta, \epsilon_{\text{GM}}, \epsilon_{\text{EM}}, \delta)$ from each party, wait further instructions from the adversary $\mathcal{A}$. If $\mathcal{A}$ inputs OK, output True to all parties and continue, otherwise output False and abort.

Model Compute: Set $\boldsymbol{S}_i = \mathtt{diag}(\mathbf{1}^n)$, for $i \in N$. For $t \in T$

1. Compute $\boldsymbol{A}_i, \boldsymbol{b}_i$ with respect to the inputs $(\boldsymbol{X}_i, \mathbf{y}_i)$ and $\boldsymbol{S}_i$, for $i \in N$ and wait further instructions from $\mathcal{A}$. If $\mathcal{A}$ inputs OK, output True to all parties and continue, otherwise output False and abort.
2. Execute a WLS iterative procedure (as described in $\Pi_{\texttt{ADMM}}$) for K steps with inputs $\boldsymbol{A}_i, \boldsymbol{b}_i$, to obtain $\bar{\mathbf{w}}^t$.
3. Apply the Gaussian Mechanism $\mathcal{N}[\epsilon_{\text{GM}}, \delta]$ to $\bar{\mathbf{w}}^t$ as $\mathcal{N}[\epsilon_{\text{GM}}, \delta]\left(\bar{\mathbf{w}}^t\right)$ and output $\mathcal{N}[\epsilon_{\text{GM}}, \delta]\left(\bar{\mathbf{w}}^t\right)$ to the adversary $\mathcal{A}$ and wait further instructions from $\mathcal{A}$. If $\mathcal{A}$ inputs OK, output $\mathcal{N}[\epsilon_{\text{GM}}, \delta]\left(\bar{\mathbf{w}}^t\right)$ to all parties, otherwise abort.
4. Compute $\mathbf{r}_i$ with respect to the inputs $(\boldsymbol{X}_i, \mathbf{y}_i)$ and $\mathcal{N}[\epsilon_{\text{GM}}, \delta]\left(\bar{\mathbf{w}}^t\right)$, for $i \in N$, and wait further instructions from $\mathcal{A}$. If $\mathcal{A}$ inputs OK, output True to all parties and continue, otherwise output False and abort.
5. Execute a differentially private quantile selection procedure (with the steps described in $\Pi^{\text{DP}}_{\texttt{qRankMPC}}$), which in each round l:
   - computes a midpoint q_l^t in an interval.
   - samples a sub-range using the exponential mechanism parametrized by ϵ_{EM}, $\mathcal{E}[\epsilon_{\text{EM}}]$, and updates the interval accordingly.

   Let q^t be the final output and $\{\mathsf{q}_l^t\}_{l \in L}$ the intermediate values. Output q^t and $\{\mathsf{q}_l^t\}_{l \in L}$ to the adversary $\mathcal{A}$ and wait further instructions from $\mathcal{A}$. If $\mathcal{A}$ inputs OK, output q^t and $\{\mathsf{q}_l^t\}_{l \in L}$ to all parties, otherwise abort.
6. Compute $\mathbf{s}_i = \left[\mathbb{1}\left(|\mathbf{r}_i| < \mathsf{q}^t\right)\right]$, for $i \in N$.
7. Set $\boldsymbol{S}_i = \mathtt{diag}(\mathbf{s}_i)$, for $i \in N$, and wait further instructions from $\mathcal{A}$. If $\mathcal{A}$ inputs OK, output True to all parties and continue, otherwise output False and abort.

Fig. 12: Ideal Functionality for TRIP*.

To model the secure computation, we use the Arithmetic Black Box (ABB) model introduced by [21]. The ABB is an ideal functionality (see Figure 13) in which parties can input private values, perform arithmetic

$\mathcal{F}_{\mathsf{AMPC}}$

Each value stored in this functionality is associated with a unique identifier that is given to all parties.

Input MPC Upon receiving (input, id, x) from a party $\mathcal{P}_i$ and (input, id) from all other parties, with id a fresh identifier, store (id, x).
Add Upon receiving $(\mathsf{add}, id_x, id_y, id_z)$ from all parties with id_x, id_y stored in memory and id_z fresh, retrieve x, y, compute $z = x + y$ and store (id_z, z).
Multiply Upon receiving $(\mathsf{mul}, id_x, id_y, id_z)$ from all parties with id_x, id_y stored in memory and id_z fresh, retrieve x, y, compute $z = x \cdot y$ and store (id_z, z).
Multiply-by-Constant Upon receiving $(\mathsf{scale}, id_x, a, id_z)$ from all parties with id_x stored in memory and id_z fresh, retrieve x, compute $z = a \cdot x$ and store (id_z, z).
Random-Sample Upon receiving $(\mathsf{random}, a, id_r)$ from all parties the functionality samples a uniform $r \to [0, a]$ and stores (id_r, r) where id_r is a fresh identifier.
Coin Upon receiving (coin, a, id_r) from all parties the functionality samples a uniform $\beta \to [0, a]$ sends it to $\mathcal{A}$ and waits $\mathcal{A}$ for further instructions. If $\mathcal{A}$ inputs OK send β to all parties otherwise abort.
MAC-Check Upon receiving (check, id_x) from all parties with id_x stored in memory, wait for the adversary. If the adversary inputs OK, return True to all parties otherwise return False and abort.
Open Upon receiving input $(\mathsf{open}, id_{val})$ from all parties, retrieve val and send it to $\mathcal{A}$ and wait $\hat{val}$ from $\mathcal{A}$. Output $\hat{val}$ to all parties. If $\mathcal{A}$ inputs OK and $val = \hat{val}$, send True to all parties otherwise return False and abort.

Fig. 13: The ideal functionality for arithmetic MPC with authentication

operations on secret-shared data, and reconstruct outputs without revealing intermediate values. The functionality has memory and supports operations such as addition and multiplication over a finite field $\mathbb{F}_p$. The ABB abstraction allows us to describe the protocol at a high level while relying on a concrete MPC protocol, such as SPDZ Overdrive, to instantiate the arithmetic operations securely.

In a similar taste we use the polynomial Zero-Knowledge and we define polyZK ideal functionality (see Figure 14). The functionality allows parties to input private values and prove satisfiability of polynomial relations over VOLE-shared data without revealing the underlying witnesses. As with the ABB abstraction, this functionality separates the high-level protocol description from the concrete proof system implementation (such as Quicksilver). For simplicity of presentation, both functionalities are defined over single values, although they naturally extend to vectorized inputs.

Theorem 2. *Let $\mathcal{F}_{\mathsf{AMPC}}$ denote the ideal functionality for authenticated arithmetic multi-party computation (realised by SPDZ/Overdrive [22, 39]), $\mathcal{F}_{\mathsf{polyZK}}$ the ideal functionality for zero-knowledge proofs of polynomial satisfiability (realised by Quicksilver [58]), $\mathcal{F}_{\mathsf{qrank}}$ the ideal functionality for the differentially-private q-ranked-element protocol $\Pi^{\mathrm{DP}}_{\mathtt{qRankMPC}}$ (realized by Figure 8). In the $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$,$\mathcal{F}_{\mathsf{qrank}}$-hybrid model, the protocol Π_{TRIP^*} securely implements $\mathcal{F}_{TRIP^*}$ (Figure 12) in the presence of a static malicious adversary $\mathcal{A}$ corrupting all but one party.*

Proof. Fix a PPT adversary $\mathcal{A}$ corrupting a set $\mathcal{C} \subsetneq [N]$ with $|\mathcal{C}| \leq N - 1$. We prove security in the hybrid model, replacing each subprotocol call with its corresponding ideal functionality. We construct S which runs the adversary $\mathcal{A}$ as a subroutine, and is given access to $\mathcal{F}_{\mathrm{TRIP}^*}$. It internally runs a copy of the protocol, together with copies of the ideal functionalities $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$,$\mathcal{F}_{\mathsf{qrank}}$ and we implicitly assume that it passes all communication between $\mathcal{A}$ and the environment.

- S receives the security parameter κ, the corrupted set $\mathcal{C}$, and internally instantiates copies of $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$, $\mathcal{F}_{\mathsf{qrank}}$. If at any point a corrupted party sends $\perp$ or a malformed message, S instructs $\mathcal{F}_{\mathrm{TRIP}^*}$ to abort.
- S chooses default inputs for honest parties — satisfying the range checks — e.g. 0. For every corrupted party $\mathcal{P}_i \in \mathcal{C}$, S receives $\mathcal{P}_i$'s effective input $(\boldsymbol{X}_i, \mathbf{y}_i)$ from the calls that $\mathcal{A}$ makes to its internal $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$, during the Initial Input Commitment phase. S reads the values stored in $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$ and forwards them as the corrupted parties' inputs to $\mathcal{F}_{\mathrm{TRIP}^*}$.
- S simulates Π_{CC} execution for each $\mathcal{P}_i, i \in [N]$ acting as a prover and the rest of the parties as verifiers as follows:

– **The prover is honest and verifier is controlled by $\mathcal{A}$.** S gets a default value r and calls its internal $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$ to input r on behalf of $\mathcal{P}_i$. Then, S outputs to $\mathcal{A}$ a default value β through its internal $\mathcal{F}_{\mathsf{AMPC}}$ (coin command). S computes $\rho = r + \sum_{k=1,l=1}^{k=m,l=d} 0\beta^k = r$ using 0 for honest party's input, runs its internal $\mathcal{F}_{\mathsf{AMPC}}$ to simulate this computation, and outputs ρ to the adversary through its internal $\mathcal{F}_{\mathsf{AMPC}}$, which to $\mathcal{A}$'s view is a random value due to the mask r. If $\mathcal{F}_{\mathsf{AMPC}}$ aborts, S instructs $\mathcal{F}_{\text{TRIP}^*}$ to abort, otherwise S continues and runs $\mathcal{F}_{\mathsf{polyZK}}$ on behalf of honest $\mathcal{P}_i$ and simulates the proof of the statement $\langle\rho\rangle - \langle r\rangle - \sum_{k=1,l=1}^{k=m,l=d} \langle 0\rangle \beta^k = 0$ to each other $\mathcal{A}$. $\mathcal{A}$ cannot distinguish between shares of 0 or random shares for $\langle r\rangle, \langle\rho\rangle$.
– **The prover is controlled by $\mathcal{A}$.** S receives $\boldsymbol{X}_{mpc}, \{\boldsymbol{X}_{zk,i'}\}_{i'\in[N]\setminus i}$ from its internal $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$ respectively. S receives r_{mpc}, r_{zk} from its internal $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$ and outputs to $\mathcal{A}$ a default value β through its internal $\mathcal{F}_{\mathsf{AMPC}}$. S computes $\rho = r_{mpc} + \sum_{k=1,l=1}^{k=m,l=d} \boldsymbol{X}_{mpc}[k][l]\beta^k$ using the adversary's input stored in $\mathcal{F}_{\mathsf{AMPC}}$ and outputs ρ to the adversary through its internal $\mathcal{F}_{\mathsf{AMPC}}$. S runs its internal $\mathcal{F}_{\mathsf{polyZK}}$ for the statement $\langle\rho\rangle_{i'} - \langle r_{zk}\rangle_{i'} - \sum_{k=1,l=1}^{k=m,l=d} \langle \boldsymbol{X}_{zk,i'}[k][l]\rangle \beta^k = 0$, $N-1$ times ($i' \in [N] \setminus i$) and outputs True to the adversary if the relation holds else outputs False and instructs $\mathcal{F}_{\text{TRIP}^*}$ to abort.
– P, V **are both honest.** S runs the protocol honestly with default inputs and all opened values are random.

– S runs its internal $\mathcal{F}_{\mathsf{polyZK}}$ and if any input fails the range checks or the consistency checks, outputs False and instructs $\mathcal{F}_{\text{TRIP}^*}$ to abort; otherwise outputs True and passes these inputs to $\mathcal{F}_{\text{TRIP}^*}$.
– For the rest of the protocol steps S receives any output from $\mathcal{F}_{\text{TRIP}^*}$ and simulates the execution either by simply calling its internal $\mathcal{F}_{\mathsf{polyZK}}$, $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{qrank}}$ or additionally following the steps described above.

Note that the protocols $\Pi_{\texttt{PrivSumRange}}$, $\Pi_{\texttt{RangeCheck}}$ make only black-box calls to the $\mathsf{input}, \mathsf{prove}, \mathsf{verify}$ commands of $\mathcal{F}_{\mathsf{polyZK}}$ so we do not need to argue about privacy separately and correctness follows trivially.

We argue that for any adversary $\mathcal{A}$ in the $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$,$\mathcal{F}_{\mathsf{qrank}}$-hybrid world, S outputs a view $\mathsf{view}_{\mathsf{ideal}}$ that is computationally indistinguishable from the hybrid-world view $\mathsf{view}_{\mathsf{hyb}}$ by $\mathcal{A}$.

– *Initial Input Commitment and Input Update.* No honest-party value is opened in these phases. The only value opened is ρ which is random to the view of the adversary since r is random if the prover is honest. If the prover is malicious the information leaked only considers his input. Indistinguishability follows directly from the security properties of $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$.
– *Model Update.* The ADMM computation runs entirely inside $\mathcal{F}_{\mathsf{AMPC}}$ with no opening; indistinguishability follows from the security of $\mathcal{F}_{\mathsf{AMPC}}$.
– *Model Reveal.* S programs the opened value to be $\mathcal{N}(\bar{\mathbf{w}}^t)$ as received from $\mathcal{F}_{\text{TRIP}^*}$. Since the Gaussian Mechanism is applied with identical parameters (Δ, σ) in both worlds, the distribution of the revealed value is identical in the real and ideal executions. Opening is then indistinguishable by the security of $\mathcal{F}_{\mathsf{AMPC}}$.
– *Participation Set Update.* For the residual commitment, again, the only value opened is ρ which is random if the prover is honest, so indistinguishability follows directly from $\mathcal{F}_{\mathsf{AMPC}}$, $\mathcal{F}_{\mathsf{polyZK}}$. For the DP quantile S programs $\mathcal{F}_{\mathsf{qrank}}$ with the outputs received from $\mathcal{F}_{\text{TRIP}^*}$; indistinguishability follows from the security of $\mathcal{F}_{\mathsf{qrank}}$. For the participation-set proofs indistinguisability follows from the security properties of $\mathcal{F}_{\mathsf{polyZK}}$.

$\mathcal{F}_{\mathsf{polyZK}}$

Each value stored in this functionality is associated with a unique identifier that is given to the relevant parties.

Input ZK Upon receiving $(\mathsf{input}_{\mathsf{zk}}, id_{zk}, w_{zk})$ from a party $\mathcal{P}_i$ and $(\mathsf{input}_{\mathsf{zk}}, id_{zk})$ from $\mathcal{P}_{i'}$, with id_{zk} a fresh identifier, store (id_{zk}, w_{zk}) .
Prove Polynomial Satisfiability Upon receiving $(\mathsf{prove}, f, id_w)$ from P and $(\mathsf{verify}, f, id_w)$ from V where id_w is present in memory, retrieve (id_w, w). Then send True to V if $f(w) = 0$ and False to V otherwise.

Fig. 14: The ideal functionality for polynomial ZK

C Detailed Protocol Description

We now provide a detailed description of each sub-routine of TRIP , following Figure 6, as well as a detailed description of the protocols we use from previous works.

C.1 Initial Input Commitment Phase

Hybrid Representation Each party shares its private input $(\boldsymbol{X}_i, \mathbf{y}_i)$ in two representations executing the following steps, as specified by $\Pi_{\texttt{HybRep}}$ (Figure 3).

1. **MPC Representation:** Each party $\mathcal{P}_i$ instantiates an MPC and generates N MAC-authenticated shares, $[\![(\boldsymbol{X}_i, \mathbf{y}_i)]\!]_{i'}$ for $i' \in [N]$, of its dataset for subsequent use in MPC.
2. **ZK Representation:** Each party $\mathcal{P}_i$ initiates $N-1$ pairwise ZK protocols with all other parties $\mathcal{P}_{i'}, i' \in [N] \setminus \{i\}$ and creates $N-1$ distinct privately authenticated secret input values $\langle (\boldsymbol{X}_i, \mathbf{y}_i) \rangle_{i'}$.
3. **Consistency Check:** Using the *Consistency Check* protocol from [15], which we denote by $\Pi_{\texttt{CC}}$, we ensure consistency between the ZK and MPC representations. Each $\mathcal{P}_i$ uses $\Pi_{\texttt{CC}}$, whenever combining ZK and MPC, to prove that both representations encode the same value. This mechanism ensures that the party cannot adaptively change its input or use inconsistent inputs during the rest of the computation.

Initial Range Checks Each party $\mathcal{P}_i$ proves that their input $(\boldsymbol{X}_i, \mathbf{y}_i)$ lies within a predefined input range appropriate for the training task (Figure 6, 0). For the ZK of our range proofs, we use the Quicksilver protocol ([58, Section 4.2]), which proves that the witness is a root of a system of polynomials; we call this protocol $\Pi_{\texttt{polyZK}}$. Specifically, the ZK computes the norm of $\|\mathbf{x}_{i,j}\|^2 = \mathbf{x}_{i,j}^\top \mathbf{x}_{i,j}$ and checks if $\|\mathbf{x}_{i,j}\|^2 < B_x^2$. Similarly, we use $\Pi_{\texttt{polyZK}}$ to check that $|y_{i,j}| < B_y$, by checking that $y_{i,j} < B_y$ and $y_{i,j} > -B_y$. The checks of the form $x \in [a, b]$ involve computing the bit decompositions of $(x-a)$ and $(b-x)$ and proving that each decomposition has the correct number of bits (see [15, Section 3.1] for a formal description). We refer to the check $x \in [a, b]$ implemented with QuickSilver as $\Pi_{\texttt{RangeCheck}}$. This protocol is identical to $\Pi_{\texttt{PrivSumRange}}$ in Figure 5, except that the result of the comparison with the range endpoints a, b is public rather than secret-shared.

C.2 Robust Model Compute Phase

Input Update During this process each party updates its summary statistics $(\boldsymbol{A}_i^t, \boldsymbol{b}_i^t)$ using Equation (2) to reflect changes instructed by the thresholding step of the previous round, by executing the following steps:

1. $\mathcal{P}_i$ securely computes $(\boldsymbol{A}_i^t, \boldsymbol{b}_i^t)$ over their sensitive data in plaintext by performing local matrix multiplications (Figure 6, 1 – 5).
2. $\mathcal{P}_i$ commits to those individual updates using Hybrid Representation (Figure 6, 1 – 5)
3. $\mathcal{P}_i$ proves correctness in ZK of those local computations (Figure 6, 1 – 5).

For the matrix multiplication proofs, we adopt a straightforward modification of the Quicksilver protocol ([58, Section 4.2]). In the original Quicksilver protocol, the output matrix is assumed to be public. Specifically, each inner product is encoded as a polynomial, and correctness is verified by checking that the polynomial evaluates to zero at the origin. However, in our setting, the output must remain private. This is straightforward to address: we introduce a variable for the constant term (i.e., the result), which we subtract from the original polynomial, and we prove that the resulting polynomial evaluates to zero. The full matrix multiplication proof reduces to multiple inner product checks, which are batched for efficiency.

Model Update This sub-routine essentially uses the ADMM optimization method inside MPC. The ADMM algorithm itself runs for K steps and for brevity, we omit the round index t, and we only keep the index k in the following description. In each step k, the following computations are carried out as described in Figure 4.

1. **Individual Model Update:** Parties jointly compute shares of the individual models $\mathbf{w}_i^k$ for $i \in [N]$ through an MPC protocol. This step performs computation involving the following quantities:
   - shares of $\mathcal{P}_i$'s summary statistics $\boldsymbol{A}_i, \boldsymbol{b}_i$;
   - shares of $\mathcal{P}_i$'s previous dual variable $\mathbf{u}_i^{k-1}$;

– the secret shared global model of the previous step in the ADMM algorithm, $\bar{\mathbf{w}}^{k-1}$.

2. **Global Model Update:** Parties compute shares of an updated global model $\bar{\mathbf{w}}^k$ by securely aggregating the newly suggested individual models from all parties. Due to the linearity of the computation, this step is performed locally using shares held by each party.
3. **Dual Variable Update:** Parties update locally their shares of $\mathcal{P}_i$'s dual variable $\mathbf{u}_i^k$ for $i \in [N]$ – using only linear computations – which captures the accumulated inconsistency between each individual model and the global model. This step ensures convergence of the ADMM algorithm by iteratively attuning discrepancies between the individual models.

Note that, all computations are carried out securely among N parties in a maliciously secure MPC using MAC-authenticated secret sharing. Intermediate values are checked using the authenticated MACs ensuring that the computations remain consistent with the private data committed in MPC during the Input Update Phase.

Model Reveal At the end of the Model Update sub-routine in Round t, parties obtain shares of the current global model $\bar{\mathbf{w}}^t = \bar{\mathbf{w}}^{t,K}$–the global model output from ADMM, after K execution steps– that represents an estimation of the optimal model parameters over the identified "corruption-free" points in the current round. The parties reconstruct the approximation model $\bar{\mathbf{w}}^t$ (7) and obtain $\bar{\mathbf{w}}^t$ in plaintext.

Participation Set Update This sub-routine updates the set of inlier data points –those with low absolute residual error under the current global model $\bar{\mathbf{w}}^t$– to be used in the next round of computation. To ensure privacy, this filtering is performed securely via the following key steps.

– **Secure Computation of each $\mathcal{P}_i$'s Residual Vector:** The process begins with the computation of the residual vector of each $\mathcal{P}_i$, which is defined as $\mathbf{r}_i^t \overset{\text{def}}{=} \boldsymbol{X}_i \cdot \bar{\mathbf{w}}^t - \mathbf{y}_i$. Since $\bar{\mathbf{w}}^t$ is publicly known, $\mathcal{P}_i$ can compute its own residual vector in plaintext (Figure 6, (8)) and each party $\mathcal{P}_{i'}$, $i' \in [N]$ can compute the share $[\![\mathbf{r}_i^t]\!]_{i'} = [\![\boldsymbol{X}_i^\top]\!]_{i'} \boxminus \bar{\mathbf{w}}^t [\![\mathbf{y}_i]\!]_{i'}$ (Figure 6, (8)). To ensure consistency between $\mathbf{r}_i^t$ and the remaining steps in the participation set update, $\mathcal{P}_i$ computes a hybrid representation for $\mathbf{r}_i^t$ by initiating a $\Pi_{\texttt{HybRep}}$ which uses the already computed MAC-authenticated shares $[\![\mathbf{r}_i^t]\!]$ instead of computing fresh shares (Figure 6, (8)).
– **Secure Computation of Quantile:** The parties jointly compute the threshold for identifying "clean" data points to be included in the next round's participation set. This is done by computing the quantile value q^t (Figure 6, (9)) of the absolute residual values using a modified version of $\Pi_{\texttt{qRankMPC}}$ (see fig. 8 without the annotations) of [2], which we describe below. The rank q is $q = (1-\beta-\eta)m$ where $\eta < 1$ is an absolute constant (e.g. $\eta = 0.1$) required by TORRENT to guarantee model recovery. Intuitively, in each iteration, slightly more points than the number of corrupted points must be filtered out.
– **Secure Update of each $\mathcal{P}_i$'s Participation Set:** Each party $\mathcal{P}_i$ computes its local participation set (Figure 6, (10)) as in Equation (1) and converts it to the binary vector $\mathbf{s}_i^{t+1}$. Each entry in the vector indicates whether the corresponding absolute residual value is below the quantile threshold q^t. The parties compute their participation matrix for the next round as $\boldsymbol{S}_i^{t+1} = \texttt{diag}(\mathbf{s}_i^{t+1})$. Additionally, each party knowing its participation vector in plaintext creates a hybrid representation for $\mathbf{s}_i^{t+1}$ using $\Pi_{\texttt{HybRep}}$ (Figure 6, (10)). We introduce a range proof with private output, detailed in Figure 5, which we refer to as $\Pi_{\texttt{PrivSumRange}}$ to prove that the participation sets $\mathbf{s}_i^t$ were correctly calculated based on the residuals and the quantile q^t (Figure 6, (10)). Essentially, $\mathcal{P}_i$ uses $\Pi_{\texttt{PrivSumRange}}$ to prove equality between the vectors $\langle \mathbf{s}_i^{t+1} \rangle_{i'}$ and the vector $\left[\langle \mathbb{1}\left(\langle r_{i,j} \rangle < \mathsf{q}^t\right) \rangle_{i'}\right]_{j\in[n]}$ for $i' \in [N] \setminus \{i\}$. Note that $\Pi_{\texttt{PrivSumRange}}$ requires $\mathcal{P}_i$ to prove the correct computation of $\left[\langle \mathbb{1}\left(\langle r_{i,j} \rangle < \mathsf{q}^t\right) \rangle_{i'}\right]_{j\in[n]}$ using its previously committed residual vector $\langle \mathbf{r}_i \rangle_{i'}$.

This sub-routine marks the end of the current round. In the following round, the protocol updates the regressor to better fit this refined participation set.

D Robust Techniques

Robust regression techniques can be broadly categorized into two classes: *thresholding*-based methods and *reweighting*-based methods.

Thresholding. This approach, used in both [24] and [8], involves iteratively identifying and removing potentially corrupted datapoints based on a scoring rule, and then refitting the regression model to the remaining dataset. A key insight behind this iterative process is that, in the early stages, it is difficult to distinguish between clean and corrupted points. This uncertainty has led to two distinct strategies for applying thresholding:

1. Iterative full-data filtering [8]: In each iteration, the model is evaluated on the entire dataset and the β fraction of points with the highest residual error under the current model is filtered-out. Then, the model is refit on the remaining points. Crucially, no points are permanently removed – each iteration starts again from the full dataset, allowing previously discarded points to potentially re-enter the model.
2. Gradual filtering [24]: In contrast, this strategy uses a filtered dataset that gradually shrinks over iterations. Over successive iterations, this leads to the removal of the whole fraction β of corrupted points.

Reweighting. Similarly, reweighting has long been a fundamental approach in statistical learning. Here, weights are used to emphasize or de-emphasize specific datapoints. A classic reweighting algorithm is the IRLS algorithm [19,41], which constructs an iterative framework from the solutions of weighted least squares, iteratively adjusting weights to converge toward the optimal ℓ_2-approximation. This principle has been lately extended in more refined algorithms such as [46, 47], where each data point is assigned *adaptively* a weight based on its likelihood of being corrupted. Over multiple iterations, inliers receive higher weights, while outliers are assigned lower weights.

E Evaluation of Robustness of TRIP*

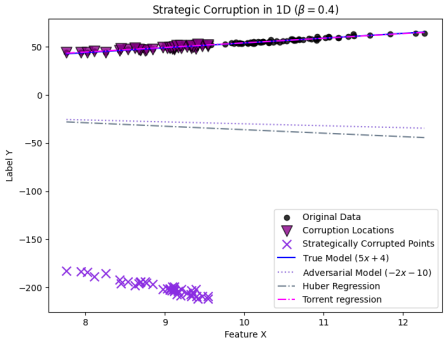


Fig. 15: $\mathcal{A}$ chooses the corruption locations in a 1D dataset of observations and replaces their target values with strategically corrupted values. The adversarial model used is fixed to $\mathbf{w}_{\mathcal{A}} = -2x - 10$.

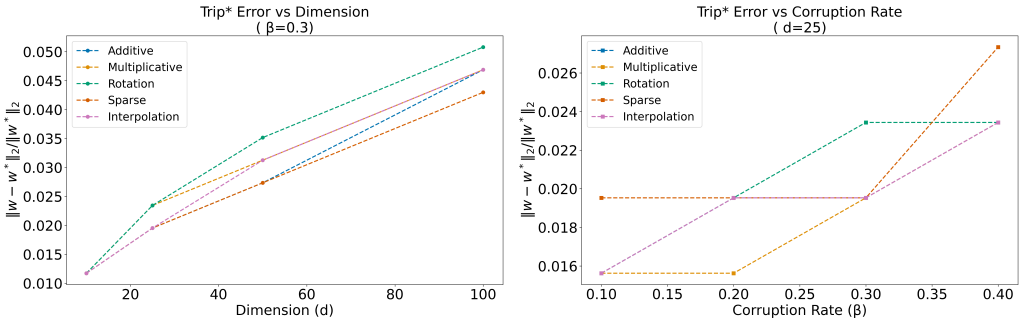


Fig. 16: TRIP* resilience with different corruption strategies for $\boldsymbol{m = 10^5}$.

Our main intuition is the following for the attack strategy: The adversary $\mathcal{A}$ inspects the input feature matrix and the response vector. Given this information, the adversary selects a set of corrupted points to manipulate in order to achieve a specific objective. The adversarial strategy proceeds as follows:

1. *Regression Coefficients Selection.* The adversary $\mathcal{A}$ chooses a regression coefficient vector $\mathbf{w}_{\mathcal{A}} \in \mathbb{R}^d$, which differs from the true regression coefficients $\mathbf{w}^* \in \mathbb{R}^d$ underlying the clean-data model fitting.
2. *Corruption Locations Selection.* The adversary then selects a subset $\mathcal{S}_{\mathcal{A}} \subset [m]$ of at most βm indices (resulting in at most βm corrupted points). The goal of $\mathcal{A}$ is to modify the responses at these points such that the predictions generated by $\mathbf{w}_{\mathcal{A}}$ are as close as possible to the true responses in the same locations.

For an example in the one-dimensional setting, we refer the reader to Figure 15, which illustrates how $\mathcal{A}$ chooses the corruption locations and the corruption values for the responses on these locations.

Given a data matrix $\boldsymbol{X} \in \mathbb{R}^{m \times d}$, where each column is a feature vector $\mathbf{x}_i$, uncorrupted labels $\mathbf{y} \in \mathbb{R}^m$, where each entry is a label y_i, an adversarial corruption model $\mathbf{w}_{\mathcal{A}} \in \mathbb{R}^d$ and a corruption fraction $\beta \in [0, 1]$, representing the fraction of response values to be corrupted, the adversary $\mathcal{A}$ initially computes the ideal response vector as:

$$\mathbf{y}_{\mathcal{A}}^{\text{ideal}} = \boldsymbol{X}\mathbf{w}_{\mathcal{A}}$$

To determine how far each sample lies from the adversarial corruption model, $\mathcal{A}$ computes the absolute deviation:

$$\delta_i = \left| y_i - y_{\mathcal{A},i}^{\text{ideal}} \right|, \quad \forall i \in \{1, \ldots, m\}$$

$\mathcal{A}$ selects the $k = \lfloor \beta m \rfloor$ samples that require the least modification, i.e., those with the smallest values of δ_i. Let $\mathcal{S}_{\mathcal{A}}$ denote the set with selected corruption locations. $\mathcal{A}$ fixes a scaling factor as $\phi = \phi_0 + \left(\frac{\phi_I}{\beta}\right)$, where ϕ_0 denotes the base scaling and ϕ_I denotes the influence factor, i.e. controls how much adversarial points are pushed in the direction of the corruption model. In practice we set, $\phi_0 = 1, \phi_I = 0.5$. For each index in $\mathcal{S}_{\mathcal{A}}$, $\mathcal{A}$ replaces the corresponding label with its scaled-adversarial counterpart:

$$\mathbf{y}_{\text{corrupted}} = \begin{cases} y_{\mathcal{A},i}^{\text{ideal}} + \phi\delta_i, & \text{if } i \in \mathcal{S}_{\mathcal{A}}, \\ y_i, & \text{otherwise.} \end{cases}$$

F Corruption Strategies

We benchmark the following corruption strategies to test the convergence speed and the convergence guarantees depending on the dimensionality and the fraction of corrupted labels. We report the results in Figure 15. For the following corruption strategies, TORRENT can resist adversarial corruptions for a fraction of corrupted labels of at most 0.4.

Arbitrary Rotation. Let $\boldsymbol{Q} \in \mathbb{R}^{d \times d}$ be an orthogonal matrix. Then the adversarial model is given by $\mathbf{w}_{\mathcal{A}} = \boldsymbol{Q}\mathbf{w}^*$.

Sparse Noise. Let $\xi \in \mathbb{R}^d$ be a random uniform noise vector and $\mathbf{b} \in \{0, 1\}^d, b_i \sim \mathcal{B}(p), i \in [d]$ a Bernoulli vector with probability of success p, each b_i indicating whether $\mathcal{A}$ applies noise or not. Then, $\mathbf{w}_{\mathcal{A}} = \mathbf{w}^* + \mathbf{b} \odot \xi$.

Interpolation between $\mathbf{w}^*$ *and a random model.* Given a mixing ratio $\lambda \in R$ and a random Gaussian model $\mathbf{w}_{\text{rand}} \sim \mathcal{N}(\mu, \boldsymbol{\Sigma})$ the corrupted model is computed as $\mathbf{w}_{\mathcal{A}} = (1 - \lambda)\mathbf{w}^* + \lambda\mathbf{w}_{\text{rand}}$.

Multiplicative/Additive corruption of $\mathbf{w}^*$. Given a parameter γ the corrupted model is computed as (1) $\mathbf{w}_{\mathcal{A}} = \gamma\mathbf{w}^*$ or (2) $\mathbf{w}_{\mathcal{A}} = \gamma + \mathbf{w}^*$.