

487 Supplementary Material

488 A Proofs for Section 5

489 Proof of Theorem 1

490 First we recall the notion of Rényi Divergences and Concentrated Differential Privacy [11, 20], as
 491 well as some other standard DP notions. We also define the Discrete Gaussian and provide its DP
 492 guarantees. See [31] for more details. Then we prove Theorem 1

493 **Definition 1** (Rényi Divergences). *Let P and Q be probability distributions on some common domain*
 494 *Ω . Assume that P is absolutely continuous with respect to Q so that the Radon-Nikodym derivative*
 495 *$P(x)/Q(x)$ is well-defined for $x \in \Omega$.*

For $\alpha \in (1, \infty)$, we define the Rényi Divergence of order α of P with respect to Q as:

$$D_\alpha(P||Q) := \frac{1}{\alpha - 1} \log \mathbb{E}_{X \leftarrow P} \left[\left(\frac{P(X)}{Q(x)} \right)^{\alpha-1} \right]$$

We also define

$$D_*(P||Q) := \sup_{\alpha \in (1, \infty)} \frac{1}{\alpha} D_\alpha(P||Q)$$

496 **Definition 2** (Concentrated Differential Privacy [11, 20]). *A randomized algorithm $M : \mathcal{X}^* \rightarrow \mathcal{Y}$*
 497 *satisfies $\frac{1}{2}\varepsilon$ -concentrated differential privacy iff, for all $x, x' \in \mathcal{X}$ differing by the addition or removal*
 498 *of a single user's records, we have $D_*(M(x)||M(x')) \leq \frac{1}{2}\varepsilon^2$.*

499 **Definition 3** (Rényi Differential Privacy [39]). *A randomized algorithm $M : \mathcal{X}^* \rightarrow \mathcal{Y}$ satisfies*
 500 *(α, ε) -Rényi differential privacy iff, for all $x, x' \in \mathcal{X}$ differing by the addition or removal of a single*
 501 *user's records, we have $D_\alpha(M(x)||M(x')) \leq \frac{1}{2}\varepsilon^2$.*

Definition 4 (Differential Privacy [18]). *A randomized algorithm $M : \mathcal{X}^* \rightarrow \mathcal{Y}$ satisfies (ε, δ) -*
differential privacy iff, for all $x, x' \in \mathcal{X}$ differing by the addition or removal of a single user's records,
we have

$$\Pr[M(x) \in E] \leq e^\varepsilon \Pr[M(x') \in E] + \delta,$$

502 *for all events $E \subset \mathcal{Y}$. We refer to $(\varepsilon, 0)$ -DP as pure DP and (ε, δ) -DP for $\delta > 0$ as approximate DP.*

503 We remark that $\frac{1}{2}\varepsilon^2$ -concentrated DP is equivalent to satisfying $(\alpha, \frac{1}{2}\varepsilon^2\alpha)$ -Rényi DP simultaneously
 504 for all $\alpha \in (1, \infty)$. We also have the following conversion lemma from concentrated to approximate
 505 DP [5, 13, 3].

Lemma 1. *If M satisfies $(\varepsilon, 0)$ -DP, then it satisfies $\frac{1}{2}\varepsilon^2$ -concentrated DP. If M satisfies $\frac{1}{2}\varepsilon^2$ -DP
 then, for any $\delta > 0$, M satisfies $(\varepsilon_{aDP}(\delta), \delta)$ -DP, where*

$$\varepsilon_{aDP}(\delta) = \inf_{\alpha > 1} \frac{1}{2}\varepsilon^2\alpha + \frac{\log(1/\alpha\delta)}{\alpha - 1} + \log(1 - 1/\alpha) \leq \varepsilon \cdot (\sqrt{2\log(1/\delta)} + \varepsilon/2).$$

506 **Discrete Gaussian** Here we define the Discrete Gaussian [13] and give its DP guarantees.

Definition 5 (Discrete Gaussian). *The discrete Gaussian with scale parameter $\sigma > 0$ and location*
parameter $\mu \in \mathbb{Z}$ is a probability distribution supported on the integers \mathbb{Z} denoted by $\mathcal{N}_{\mathbb{Z}}(\mu, \sigma^2)$ and
defined by

$$\forall x \in \mathbb{Z} \quad \Pr_{X \leftarrow \mathcal{N}_{\mathbb{Z}}(\mu, \sigma^2)}(X = x) = \frac{\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)}{\sum_{y \in \mathbb{Z}} \exp\left(\frac{-(y-\mu)^2}{2\sigma^2}\right)}.$$

507 **Proposition 1** ([31], Proposition 14). *Let $\sigma \geq \frac{1}{2}$. Let $X_{i,j} \leftarrow \mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$ independently for each i*
 508 *and j . Let $X_i = (X_{i,1}, \dots, X_{i,d}) \in \mathbb{Z}^d$. Let $Z_n = \sum_{i=1}^n X_i \in \mathbb{Z}^d$. Then, for all $\Delta \in \mathbb{Z}^d$ and all*

509 $\alpha \in (1, \infty)$,

$$D_\alpha(Z_n \| Z_n + \Delta) \leq \min \left\{ \frac{\alpha \|\Delta\|_2^2}{2n\sigma^2} + \tau d, \right. \\ \left. \frac{\alpha}{2} \cdot \left(\frac{\|\Delta\|_2^2}{n\sigma^2} + 2 \frac{\|\Delta\|_1}{\sqrt{n}\sigma} \cdot \tau + \tau^2 d \right), \right. \\ \left. \frac{\alpha}{2} \cdot \left(\frac{\|\Delta\|_2}{\sqrt{n}\sigma} + \tau \sqrt{d} \right)^2 \right\}$$

510 where $\tau := 10 \cdot \sum_{k=1}^n e^{-2\pi^2 \sigma^2 \frac{k}{k+1}}$. An algorithm M that adds Z_n to a query with ℓ_p sensitivity Δ_p
 511 satisfies $\frac{1}{2}\varepsilon^2$ -concentrated DP for

$$\varepsilon = \min \left\{ \sqrt{\frac{\|\Delta\|_2^2}{n\sigma^2} + 2\tau d}, \right. \\ \left. \sqrt{\frac{\Delta_2^2}{n\sigma^2} + 2 \frac{\Delta_1}{\sqrt{n}\sigma} \cdot \tau + \tau^2 d}, \right. \\ \left. \frac{\Delta_2}{\sqrt{n}\sigma} + \tau \sqrt{d} \right\}$$

512 Proof of Theorem 1

Proof. First, it is sufficient to show that the computation $\mathbf{C}\mathbf{G} + \mathbf{Z}$ satisfies $\frac{1}{2}\varepsilon^2$ -concentrated DP, due to the post processing property of DP. Now consider two datasets \mathbf{G} and \mathbf{H} differing in one data record according to participation schema Φ .⁴ By assumption in the theorem statement, we have

$$\text{sens}_\Phi(\mathbf{C}) = \Delta.$$

513 With the bound on the total sensitivity above, we know from [31, Proposition 14] (reproduced above)
 514 that the computation is $\frac{1}{2}\varepsilon^2$ -concentrated DP, with the ε from the theorem statement. \square

515 Proof of Theorem 2

516 We first prove the following exact result for the error:

Theorem 3.

$$\hat{\sigma}^2(x) := \frac{\rho \cdot \|\mathbf{A}_{[T,:]\|_2^2}{d} \sum_{\tau=1}^T \sum_{i=1}^n \|\mathbf{g}_{\tau,i}\|_2^2 + \left(\frac{\gamma^2 \cdot \|\mathbf{A}_{[T,:]\|_2^2}{4} + \sigma^2 \cdot \|\mathbf{B}_{[T,:]\|_2^2} \right) \cdot n \\ \leq \frac{\rho \|\mathbf{A}_{[T,:]\|_2^2}{d} c^2 n T + \left(\frac{\gamma^2 \cdot \|\mathbf{A}_{[T,:]\|_2^2}{4} + \|\mathbf{B}\|_2^2 \cdot \sigma^2 \right) \cdot n$$

517 If $\hat{\sigma}^2(x) \leq r^2$ then

$$\mathbb{E}[\|\mathcal{A}(x) - \mathbf{A}_{[T,:]\|_2^2 \left(\sum_{i=1}^n \mathbf{x}_i \right)\|_2^2] \leq \frac{dn}{1-\beta} \left(\frac{2\sqrt{2} \cdot r \cdot e^{-r^2/4\hat{\sigma}^2(x)}}{\sqrt{n(1-\beta)^{nT-1}}} \right. \\ \left. + \left(\|\mathbf{A}_{[T,:]\|_2^2 \cdot \left(\frac{\gamma^2}{4} + \frac{\beta^2 \cdot \gamma^2 n}{1-\beta} \right) + \|\mathbf{B}_{[T,:]\|_2^2 \cdot \sigma^2 \right)^{1/2} \right)^2.$$

518 We start with a modified version of Proposition 26 in [31].

Proposition 2.

$$\mathbb{E}[\|\mathcal{A}(x) - \mathbf{A}_{[T,:]\|_2^2 \sum_{i=1}^n \mathbf{X}_i\|_2^2] \leq \|\mathbf{A}_{[T,:]\|_2^2 \cdot \left(\frac{\gamma^2 \cdot d \cdot n}{4(1-\beta)} + \left(\frac{\beta}{1-\beta} \gamma \sqrt{dn} \right)^2 \right) + \|\mathbf{B}_{[T,:]\|_2^2 \cdot n \cdot d \cdot \sigma^2.$$

519

$$\forall \mathbf{t} \in \mathbb{R}^d \quad \mathbb{E}[\exp(\langle \mathbf{t}, \mathcal{A}(x) - \mathbf{A}_{[T,:]\|_2^2 \sum_{i=1}^n \mathbf{X}_i \rangle)] \leq \frac{\exp\left(\left(\frac{\gamma^2 \cdot \|\mathbf{A}_{[T,:]\|_2^2}{8} + \frac{\sigma^2 \cdot \|\mathbf{B}_{[T,:]\|_2^2}{2}\right) \cdot \|\mathbf{t}\|_2^2 \cdot n\right)}{(1-\beta)^{nT}}.$$

⁴ \mathbf{G} and \mathbf{H} really consist of entries that are sums of records.

520 *Proof.* We have

$$\begin{aligned}
\mathbb{E} \left[\left\| \mathcal{A}(x) - \mathbf{A}_{[T,:]} \sum_{i=1}^n \mathbf{X}_i \right\|_2^2 \right] &= \mathbb{E} \left[\left\| \sum_{\tau=1}^T \mathbf{A}_{T,\tau} \cdot \left(\sum_{i=1}^n (R_\gamma^G(\mathbf{g}_{\tau,i}) - \mathbf{g}_{\tau,i}) \right) + \mathbf{B}_{T,\tau} \cdot \sum_{i=1}^n \gamma \cdot \mathbf{z}_{\tau,i} \right\|_2^2 \right] \\
&\leq \sum_{\tau=1}^T \mathbf{A}_{T,\tau}^2 \cdot \mathbb{E} \left[\left\| \sum_{i=1}^n R_\gamma^G(\mathbf{g}_{\tau,i}) - \mathbf{g}_{\tau,i} \right\|_2^2 \right] + \mathbf{B}_{T,\tau}^2 \cdot n \cdot \sigma^2 \\
&\leq \|\mathbf{A}_{[T,:]\|_2^2 \cdot \left(\frac{\gamma^2 \cdot d \cdot n}{4(1-\beta)} + \left(\frac{\beta}{1-\beta} \gamma \sqrt{dn} \right)^2 \right) + \|\mathbf{B}_{[T,:]\|_2^2 \cdot n \cdot \sigma^2,
\end{aligned}$$

521 where the last inequality is due directly to Proposition 26 of [31].

522 Now, for each $i \in [n], \tau \in [T]$, we have that $R_\gamma(\mathbf{g}_{\tau,i}) \in \gamma \lfloor \mathbf{g}_{\tau,i} / \gamma \rfloor + \{0, \gamma\}^d$ and is a product
523 distribution with mean $\mathbf{g}_{\tau,i}$. Thus, $R_\gamma(\mathbf{g}_{\tau,i}) - \mathbf{g}_{\tau,i} \in \{0, \gamma\}^d$ and is a product distribution with mean
524 $\mathbf{0}$. Therefore, by Hoeffding's lemma, we have:

$$\forall \mathbf{t} \in \mathbb{R}^d \quad \mathbb{E}[\exp(\langle \mathbf{t}, \sum_{\tau=1}^T \mathbf{A}_{T,\tau} \sum_{i=1}^n R_\gamma(\mathbf{g}_{\tau,i}) - \mathbf{g}_{\tau,i} \rangle)] \leq \exp\left(\frac{\gamma^2}{8} \cdot n \cdot \|\mathbf{A}_{[T,:]\|_2^2 \cdot \|\mathbf{t}\|_2^2\right).$$

525 Thus,

$$\begin{aligned}
\forall \mathbf{t} \in \mathbb{R}^d \quad \mathbb{E}[\exp(\langle \mathbf{t}, \sum_{\tau=1}^T \mathbf{A}_{T,\tau} \sum_{i=1}^n R_\gamma^G(\mathbf{g}_{\tau,i}) - \mathbf{g}_{\tau,i} \rangle)] &\leq \frac{\mathbb{E}[\exp(\langle \mathbf{t}, \sum_{\tau=1}^T \mathbf{A}_{T,\tau} \sum_{i=1}^n R_\gamma(\mathbf{g}_{\tau,i}) - \mathbf{g}_{\tau,i} \rangle)]}{\Pr[R_\gamma(\mathbf{g}_{\tau,i}) \in G \forall \tau, i]} \\
&\leq \frac{\exp(\frac{\gamma^2}{8} \cdot n \cdot \|\mathbf{A}_{[T,:]\|_2^2 \cdot \|\mathbf{t}\|_2^2)}{(1-\beta)^{nT}}.
\end{aligned}$$

526 Moreover, we have that [13]:

$$\forall \mathbf{t} \in \mathbb{R}^d \quad \mathbb{E}[\exp(\langle \mathbf{t}, \sum_{\tau=1}^T \mathbf{B}_{T,\tau} \sum_{i=1}^n \gamma \cdot \mathbf{z}_{\tau,i} \rangle)] \leq \exp\left(\frac{\sigma^2}{2} \cdot n \cdot \|\mathbf{B}_{[T,:]\|_2^2 \cdot \|\mathbf{t}\|_2^2\right).$$

527

□

528 Finally, we are able to prove a modified version of Theorem 36 from [31].

529 *Proof of Theorem 3.* By assumption, we have that

$$\forall \mathbf{x} \in \mathbb{R}^d \forall j \in [d] \forall t \in \mathbb{R} \quad \mathbb{E}[\exp(t(\mathbf{U}\mathbf{x})_j)] \leq \exp(t^2 \rho \|\mathbf{x}\|_2^2 / 2d).$$

530 Therefore,

$$\begin{aligned}
\mathbb{E}[\exp(t \cdot (\sum_{\tau=1}^T \mathbf{A}_{T,\tau} \cdot (\mathbf{U} \sum_{i=1}^n \mathbf{g}_{\tau,i})_j))] &= \prod_{\tau=1}^T \cdot \prod_{i=1}^n \mathbb{E}[\exp(t \cdot \mathbf{A}_{T,\tau} \cdot (\mathbf{U} \mathbf{g}_{\tau,i})_j)] \\
&\leq \prod_{\tau=1}^T \cdot \prod_{i=1}^n \exp(t^2 \cdot \mathbf{A}_{T,\tau}^2 \cdot \rho \cdot \|\mathbf{g}_{\tau,i}\|_2^2 / 2d) \\
&= \exp(t^2 \cdot \|\mathbf{A}_{[T,:]\|_2^2 \cdot \rho \cdot \sum_{\tau=1}^T \sum_{i=1}^n \|\mathbf{g}_{\tau,i}\|_2^2 / 2d).
\end{aligned}$$

531 Combining with the result of Proposition 2, we have

$$\begin{aligned}
\forall t \in \mathbb{R} \forall j \in [d] \quad \mathbb{E}[\exp(t \cdot (\mathcal{A}(\mathbf{U}\mathbf{x}))_j)] &\leq \exp\left(\frac{t^2 \cdot \|\mathbf{A}_{[T,:]\|_2^2 \cdot \rho}{2d} \cdot \sum_{\tau=1}^T \sum_{i=1}^n \|\mathbf{g}_{\tau,i}\|_2^2\right) \\
&\quad \cdot \frac{\exp\left(\left(\frac{\gamma^2 \cdot \|\mathbf{A}_{[T,:]\|_2^2}{8} + \frac{\sigma^2 \cdot \|\mathbf{B}_{[T,:]\|_2^2}{2}\right) \cdot t^2 \cdot n\right)}{(1-\beta)^{nT}}
\end{aligned}$$

532 Recall $\hat{\sigma}^2(x) = \frac{\rho \cdot \|\mathbf{A}_{[T,:]\|_2^2}{d} \sum_{\tau=1}^T \sum_{i=1}^n \|\mathbf{g}_{\tau,i}\|_2^2 + (\frac{\gamma^2 \cdot \|\mathbf{A}_{[T,:]\|_2^2}{4} + \sigma^2 \cdot \|\mathbf{B}_{[T,:]\|_2^2}) \cdot n$.

533 By Proposition 35 of [31], for all $j \in [d]$,

$$\mathbb{E}[(M_{[a,b]}(\mathcal{A}(\mathbf{U}x))_j - \mathcal{A}(\mathbf{U}x)_j)^2] \leq (b-a)^2 \cdot \frac{1}{(1-\beta)^{nT}} \cdot e^{-(b-a)^2/8\hat{\sigma}^2(x)} \cdot (e^{\frac{a^2-b^2}{4\hat{\sigma}^2}} + e^{\frac{b^2-a^2}{4\hat{\sigma}^2}}),$$

534 where $a = -r$ and $b = r$ here. Summing over $j \in [d]$ gives

$$\mathbb{E}[\|M_{[-r,r]}(\mathcal{A}(\mathbf{U}x)) - \mathcal{A}(\mathbf{U}x)\|_2^2] \leq 4r^2 \cdot \frac{d}{(1-\beta)^{nT}} \cdot e^{-r^2/2\hat{\sigma}^2(x)} \cdot 2$$

535 Continuing with the proof from [31], we get:

$$\begin{aligned} & \mathbb{E}[\|\tilde{\mathcal{A}}(x) - \mathbf{A}_{[T,:]\sum_{i=1}^n \mathbf{X}_i\|_2^2] \\ & \leq \left((8r^2 \cdot \frac{d}{(1-\beta)^{nT}} \cdot e^{-r^2/2\hat{\sigma}^2(x)})^{1/2} + \left(\|\mathbf{A}_{[T,:]\|_2^2 \cdot \left(\frac{\gamma^2 \cdot d \cdot n}{4(1-\beta)} + \left(\frac{\beta}{1-\beta} \gamma \sqrt{dn} \right)^2 \right) + \right. \right. \\ & \quad \left. \left. \|\mathbf{B}_{[T,:]\|_2^2 \cdot n \cdot d \cdot \sigma^2 \right)^{1/2} \right)^2 \\ & = \frac{dn}{1-\beta} \left(\frac{2\sqrt{2} \cdot r \cdot e^{-r^2/4\hat{\sigma}^2(x)}}{\sqrt{n(1-\beta)^{nT-1}}} + \left(\|\mathbf{A}_{[T,:]\|_2^2 \cdot \left(\frac{\gamma^2}{4} + \frac{\beta^2 \cdot \gamma^2 n}{1-\beta} \right) + \|\mathbf{B}_{[T,:]\|_2^2 \cdot \sigma^2 \right)^{1/2} \right)^2. \end{aligned}$$

536 □

537 With this error bound, assuming that $\beta \leq 1/\sqrt{n}$ and $\hat{\sigma}^2(x) \leq r^2/4 \log(r\sqrt{n}/\gamma^2)$, we get

$$\mathbb{E}[\|\tilde{\mathcal{A}}(x) - \mathbf{A}_{[T,:]\sum_{i=1}^n \mathbf{X}_i\|_2^2] \leq O(dn((\|\mathbf{A}_{[T,:]\|_2^2 \cdot \gamma^2 + \|\mathbf{B}_{[T,:]\|_2^2 \cdot \sigma^2))).$$

538 *Proof of Theorem 2.* Note that $r = \frac{1}{2}\gamma m$. We verify that setting the parameters as specified yields
539 $\frac{1}{2}\varepsilon^2$ -concentrated DP and the desired accuracy. First,

$$\varepsilon^2 \leq \frac{\Delta^2 \hat{c}^2}{n\sigma^2} + 2\tau d \leq \frac{\Delta^2(c + \gamma\sqrt{d})^2}{n\sigma^2} + 20nde^{-\pi^2(\sigma/\gamma)^2} \leq \frac{2\Delta^2 c^2}{n\sigma^2} + \frac{2d\Delta^2}{n(\sigma/\gamma)^2} + 20nde^{-\pi^2(\sigma/\gamma)^2}.$$

540 Thus the privacy requirement is satisfied as long as $\sigma \geq 2c\Delta/\varepsilon\sqrt{n}$ and $(\sigma/\gamma)^2 \geq 8d\Delta^2/\varepsilon^2 n$, and
541 $20nde^{-\pi^2(\sigma/\gamma)^2} \leq \varepsilon^2/4$. So we can set

$$\sigma = \max\left\{ \frac{2c\Delta}{\varepsilon\sqrt{n}}, \frac{\gamma\Delta\sqrt{8d}}{\varepsilon\sqrt{n}}, \frac{\gamma}{\pi^2} \log\left(\frac{80nd}{\varepsilon^2}\right) \right\} = \tilde{\Theta}\left(\frac{c\Delta}{\varepsilon\sqrt{n}} + \sqrt{\frac{d}{n}} \cdot \frac{\gamma\Delta}{\varepsilon} + \gamma \log\left(\frac{nd}{\varepsilon^2}\right)\right).$$

542 We set $\beta = \min\{1/n, 1/2\} = \Theta(\frac{1}{n})$.

543 Next,

$$\begin{aligned} \hat{\sigma}^2 & \leq \frac{\rho \|\mathbf{A}_{[T,:]\|_2^2}{d} c^2 nT + \left(\frac{\gamma^2 \|\mathbf{A}_{[T,:]\|_2^2}{4} + \sigma^2 \|\mathbf{B}_{[T,:]\|_2^2} \right) \cdot n \\ & \leq \frac{\rho \|\mathbf{A}_{[T,:]\|_2^2}{d} c^2 nT + \gamma^2 \|\mathbf{A}_{[T,:]\|_2^2 n + \sigma^2 \|\mathbf{B}_{[T,:]\|_2^2 \cdot n \\ & \leq O\left(\frac{\rho \|\mathbf{A}_{[T,:]\|_2^2}{d} c^2 nT + \gamma^2 \|\mathbf{A}_{[T,:]\|_2^2 n + \|\mathbf{B}_{[T,:]\|_2^2 \left(\frac{c^2 \Delta^2}{\varepsilon^2} + \frac{\gamma^2 d \Delta}{\varepsilon^2} + \gamma^2 n \log^2\left(\frac{nd}{\varepsilon^2}\right)\right)\right) \\ & \leq O\left(\frac{\rho \|\mathbf{A}_{[T,:]\|_2^2}{d} c^2 nT + \|\mathbf{B}_{[T,:]\|_2^2 \frac{c^2 \Delta^2}{\varepsilon^2}\right) + \gamma^2 \cdot O(\|\mathbf{A}_{[T,:]\|_2^2 n + \|\mathbf{B}_{[T,:]\|_2^2 \left(\frac{d \Delta}{\varepsilon^2} + n \log^2\left(\frac{nd}{\varepsilon^2}\right)\right)). \end{aligned}$$

544 Now we work out the asymptotics of the accuracy guarantee:

$$\begin{aligned}
& \mathbb{E}[\|\tilde{\mathcal{A}}(x) - \mathbf{A}_{[T,:]} \sum_{i=1}^T \mathbf{X}_i\|_2^2] \\
& \leq \frac{dn}{1-\beta} \left(\frac{2\sqrt{2} \cdot r \cdot e^{-r^2/4\hat{\sigma}^2}(x)}{\sqrt{n(1-\beta)^{nT-1}}} + \left(\|\mathbf{A}_{[T,:]} \|_2^2 \cdot \left(\frac{\gamma^2}{4} + \frac{\beta^2 \cdot \gamma^2 n}{1-\beta} \right) + \|\mathbf{B}_{[T,:]} \|_2^2 \cdot \sigma^2 \right)^{1/2} \right)^2 \\
& \leq O(nd(\frac{re^{-r^2/4\hat{\sigma}^2}}{\sqrt{n}} + \sqrt{\|\mathbf{A}_{[T,:]} \|_2^2 \gamma^2 + \|\mathbf{B}_{[T,:]} \|_2^2 \sigma^2})) \\
& \leq O(nd(\frac{r^2 e^{-r^2/2\hat{\sigma}^2}}{n} + \|\mathbf{A}_{[T,:]} \|_2^2 \gamma^2 + \|\mathbf{B}_{[T,:]} \|_2^2 \sigma^2)) \\
& \leq O(nd(\frac{\gamma^2 m^2}{n} \exp(\frac{-\gamma^2 m^2}{8\hat{\sigma}^2}) + \|\mathbf{A}_{[T,:]} \|_2^2 \gamma^2 + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{c^2 \Delta^2}{\varepsilon^2 n} + \frac{d\gamma^2 \Delta^2}{\varepsilon^2 n} + \gamma^2 \log^2(\frac{nd}{\varepsilon^2})))) \\
& \leq O(\|\mathbf{B}_{[T,:]} \|_2^2 \frac{c^2 \Delta^2 d}{\varepsilon^2} + \gamma^2 nd(\frac{m^2}{n} \exp(\frac{-\gamma^2 m^2}{8\hat{\sigma}^2}) + \|\mathbf{A}_{[T,:]} \|_2^2 + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{d\Delta^2}{\varepsilon^2 n} + \log^2(\frac{nd}{\varepsilon^2}))))
\end{aligned}$$

545 Similarly to the analysis of Theorem 2 in [31], if

$$\begin{aligned}
m^2 & \geq O((\|\mathbf{A}_{[T,:]} \|_2^2 n + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{d\Delta}{\varepsilon^2} + n \log^2(\frac{nd}{\varepsilon^2}))) \cdot \log(1 + m^2/n)) \\
& = \tilde{O}(\|\mathbf{A}_{[T,:]} \|_2^2 n + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{d\Delta}{\varepsilon^2} + n)),
\end{aligned}$$

546 then we can set

$$\gamma^2 = O(\frac{\rho \|\mathbf{A}_{[T,:]} \|_2^2 c^2 n T}{d} + \frac{\|\mathbf{B}_{[T,:]} \|_2^2 c^2 \Delta^2}{\varepsilon^2}) \cdot \frac{\log(1 + m^2/n)}{m^2}$$

547 so that $\frac{m^2}{n} \exp(\frac{-\gamma^2 m^2}{8\hat{\sigma}^2}) \leq 1$.

548 This gives us,

$$\begin{aligned}
& \mathbb{E}[\|\tilde{\mathcal{A}}(x) - \mathbf{A}_{[T,:]} \sum_{i=1}^T \mathbf{X}_i\|_2^2] \\
& \leq O(\|\mathbf{B}_{[T,:]} \|_2^2 \frac{c^2 \Delta^2 d}{\varepsilon^2} + \gamma^2 nd(1 + \|\mathbf{A}_{[T,:]} \|_2^2 + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{d\Delta^2}{\varepsilon^2 n} + \log^2(\frac{nd}{\varepsilon^2})))) \\
& \leq O(\|\mathbf{B}_{[T,:]} \|_2^2 \frac{c^2 \Delta^2 d}{\varepsilon^2} + (\frac{\rho \|\mathbf{A}_{[T,:]} \|_2^2 c^2 n T}{d} + \frac{\|\mathbf{B}_{[T,:]} \|_2^2 c^2 \Delta^2}{\varepsilon^2}) \cdot \frac{\log(1 + m^2/n)}{m^2} nd(1 + \|\mathbf{A}_{[T,:]} \|_2^2 + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{d\Delta^2}{\varepsilon^2 n} + \log^2(\frac{nd}{\varepsilon^2})))) \\
& \leq O(\|\mathbf{B}_{[T,:]} \|_2^2 \frac{c^2 \Delta^2 d}{\varepsilon^2} + \|\mathbf{B}_{[T,:]} \|_2^2 \frac{c^2 \Delta^2 d}{\varepsilon^2} (\frac{\log(1 + m^2/n)}{m^2} n \cdot (\rho \|\mathbf{A}_{[T,:]} \|_2^2 T + 1 + \|\mathbf{A}_{[T,:]} \|_2^2 + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{d\Delta^2}{\varepsilon^2 n} + \log^2(\frac{nd}{\varepsilon^2})))))) \\
& \leq O(\|\mathbf{B}_{[T,:]} \|_2^2 \frac{c^2 \Delta^2 d}{\varepsilon^2} (1 + \frac{\log(1 + m^2/n)}{m^2} n \cdot (\rho \|\mathbf{A}_{[T,:]} \|_2^2 T + 1 + \|\mathbf{A}_{[T,:]} \|_2^2 + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{d\Delta^2}{\varepsilon^2 n} + \log^2(\frac{nd}{\varepsilon^2}))))).
\end{aligned}$$

549 So, if

$$\begin{aligned}
m^2 & \geq O(\log(1 + m^2/n) n \cdot (\rho \|\mathbf{A}_{[T,:]} \|_2^2 T + 1 + \|\mathbf{A}_{[T,:]} \|_2^2 + \|\mathbf{B}_{[T,:]} \|_2^2 (\frac{d\Delta^2}{\varepsilon^2 n} + \log^2(\frac{nd}{\varepsilon^2})))) \\
& = \tilde{O}(\rho \|\mathbf{A}_{[T,:]} \|_2^2 n T + \|\mathbf{B}_{[T,:]} \|_2^2 \frac{d\Delta^2}{\varepsilon^2}),
\end{aligned}$$

550 then the mean squared error is $O(\|\mathbf{B}_{[T,:]} \|_2^2 \frac{c^2 \Delta^2 d}{\varepsilon^2})$, as required. The final bound is obtained by simply
551 summing the above over each round from $T = 1$ to $T = T^*$. \square

B Resharing Security Model and Proof

Security proofs

We first provide an intuition on the current analysis for proving the security of cryptographic protocols. In the security proof, we compare between an n -party function $f(x_1, \dots, x_n) = (y_1, \dots, y_n)$ and a protocol $P(x_1, \dots, x_n)$ that allegedly privately computes the function f . Intuitively, a protocol P correctly and privately computes f if the following hold: (a) *Correctness*: For every input $\vec{x} = (x_1, \dots, x_n)$, the output of the parties at the end of the protocol interaction P is the same as $f(\vec{x})$; (b) *Privacy*: There exists a simulator \mathcal{S} that receives the input and output of the corrupted parties, and can efficiently generate the messages that the corrupted parties received during the protocol execution. The simulator does not know the input/outputs of the honest parties. Intuitively, the fact that the messages sent by the honest parties can be generated from the input/output of the corrupted parties implies that these messages do not contain any additional information about the inputs of the honest parties besides what is revealed from the output of the computation.

Security Model

We now introduce the formal security model. We first note that we consider robustness checks on inputs out of the scope of our security model; i.e., we do not cover *poisoning attacks*, which have been extensively studied in the literature, e.g., [46, 22]. Indeed, it is the case that malicious parties can input to the protocol whatever they want as their gradients and noise \mathbf{x}, \mathbf{z} , which can lead to a meaningless model.

We follow the standard real/ideal world security paradigm of [26]. Consider some multi-party protocol Π that is executed by some parties P_1, \dots, P_N that are grouped into committees $\mathcal{C}_1, \dots, \mathcal{C}_{T^*}$ from round 1 to round T^* and a server S . Note: the committees can be arbitrarily chosen, but our protocol only provides security if the assumption that the number of parties \mathcal{A} corrupts is at most t holds; in other words, we abstract out the committee selection process.⁵ Each of these parties has inputs $\mathbf{x}_1, \dots, \mathbf{x}_N$, and they want to evaluate some given *functionality* \mathcal{F} . In our case, the functionality $\mathcal{F}_{\text{PPFL}}$ is resharing the inputs from all previous committees to the next committee, in each round, and then outputting the \widehat{AX}_T value to the server in each round T , given some factorization $\mathbf{A} = \mathbf{BC}$. The security of protocol Π is defined by comparing the real-world execution of the protocol with an *ideal-world* evaluation of \mathcal{F} by a trusted party (ideal functionality), who receives the inputs $\mathbf{x}_1, \dots, \mathbf{x}_N$ from the parties in the clear and simply sends the relevant parties their outputs $\mathcal{F}(\mathbf{x}_1, \dots, \mathbf{x}_N)$ periodically. There is an adversary \mathcal{A} that chooses to corrupt at most $t < N$ of the parties P_1, \dots, P_N . This adversary \mathcal{A} sees all of the messages and inputs and outputs of the corrupted parties and is allowed to act arbitrarily on their behalf. We also assume that the server is corrupted and thus \mathcal{A} can see all of the messages sent to the server and all of its outputs. Informally, it is required that for every adversary that corrupts some parties during the protocol execution, there is an adversary \mathcal{S} , also referred to as the *simulator*, which can achieve the same effect and learn the same information in the ideal-world. This simulator only sees what the corrupted parties send to the honest parties and the output \mathbf{y} vectors, not the inputs \mathbf{x} of the honest parties. We now formally describe the security definition.

Real Execution. In the real execution, Π is executed in the presence of the adversary \mathcal{A} . The *view* of a party P during an execution of Π , denoted by View_P^Π consists of the messages P receives from the other parties during the execution and P 's input. The execution of Π in the presence of \mathcal{A} on inputs $(\mathbf{x}_1, \dots, \mathbf{x}_N)$ denoted $\text{Real}_{\Pi, \mathcal{A}}(\mathbf{x}_1, \dots, \mathbf{x}_N)$ is defined as $\{\text{View}_P^\Pi\}_{P \in \mathcal{C}}$. The output of Π in the presence of \mathcal{A} on inputs $(\mathbf{x}_1, \dots, \mathbf{x}_N)$ is noted as *Output*.

Ideal Execution. In the ideal execution, the parties and an ideal world adversary \mathcal{S} interact with a trusted party (ideal functionality). The ideal execution proceeds as follows: As a committee \mathcal{C}_T comes online, the parties $P_{T,1}, \dots, P_{T,n}$ in that committee send their inputs $\mathbf{x}_{T,1}, \dots, \mathbf{x}_{T,n}$ to the trusted party, who computes the output $\mathcal{F}(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{T,n})$ to the server for that round. \mathcal{S} is also allowed to release a vector χ , which will be added to the output, to simulate additive attacks.

⁵In practice, the committee selection is done by the server.

Definition 6. Protocol Π securely computes \mathcal{F} if for every adversary \mathcal{A} there exists a simulator \mathcal{S} such that

$$\text{SD}((\{\text{View}_P^\Pi\}_{P \in \mathcal{C}}, \text{Output}), (\mathcal{S}(\{\mathbf{x}_{T^*,j}\}_{T,j \in \mathcal{C}(T)}, \mathcal{F}(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{T^*,n}), \mathcal{F}(\mathbf{x}_{1,1}, \dots, \mathbf{x}_{T^*,n}) + \chi)) \leq \text{negl}(\lambda),^6$$

where SD is the statistical distance between the two distributions and $\mathcal{C}(T)$ is the set of corrupted parties in round T .

Additional Protocol Details for Active Security

Before proving the security of our protocol, we provide additional details that are needed for an adversary that is allowed to act arbitrarily on behalf of the corrupted parties, or an *active* adversary. For active security, our protocol relies on three main techniques:

1. **Commitments:** Commitments are a two-stage protocol where first a party P_i commits to some value x by using $c \leftarrow \text{Comm}(x)$ and sending c to the other parties. The important property is that $\text{Comm}(x)$ *hides* x from the other parties. Next, the party P_i can open c by using $o \leftarrow \text{Open}(c, x)$ and sending (o, x) to the other parties. The important property is that P_i cannot convince the other parties that it committed to another value $x' \neq x$ in its original commitment c . There are several well-known constructions of commitments.
2. **Random Linear Combinations:** If $\beta \in \mathbb{F}$ is random and unknown to all, then to check that some secret sharings $\text{Share}(\delta_j)$ for $j \in [n - d - 1]$ each share $\mathbf{0}$, we can compute and reconstruct $\text{Share}(\delta_j) \leftarrow \sum_{j=1}^{n-d-1} \beta^j \cdot \text{Share}(\delta_j)$, then check that the reconstructed value is $\mathbf{0}$. Intuitively, we are evaluating the polynomial defined by the δ_j on random point β . So if some $\delta_j \neq \mathbf{0}$, then by the Schwartz-Zippel Lemma, the reconstructed value will be non-zero with high probability.
3. **Parity Check Matrices:** We let $\mathbf{H} \in \mathbb{F}^{(n-\deg-1) \times n}$ be the parity check matrix such that $\mathbf{H} \cdot \mathbf{x} = \mathbf{0}$ if and only if $\mathbf{x} \in \mathbb{F}^n$ are valid shares of a polynomial of degree $\leq \deg$. This matrix intuitively takes the first $\deg + 1$ shares in \mathbf{x} , computes the other $n - (\deg + 1)$ shares (using lagrange interpolation), and compares them to those that are actually in \mathbf{x} .

With these tools in hand, we can describe the modifications to our passively-secure protocol above, to make it actively secure. After committee \mathcal{C}_{T+1} receives the shared $\text{Reshare}(\zeta_1^i, \dots, \zeta_k^i)$ from each P_i in committee \mathcal{C}_T , each party P_j in committee \mathcal{C}_{T+1} samples random β_j , sends $c \leftarrow \text{Comm}(\beta_j)$ to the other parties of committee \mathcal{C}_{T+1} and finally opens β_j to the other parties. The parties of \mathcal{C}_{T+1} then define m to be the number of parties from \mathcal{C}_T that actually sent them reshared values and compute

$$(\mathbf{y}_1^j, \dots, \mathbf{y}_{m-\deg-1}^j) \leftarrow \mathbf{H} \cdot (Z_1^j, \dots, Z_m^j).$$

Note that since the secret sharing is linear, by the properties of parity check matrices above, the shared \mathbf{y}_l will be equal to $\mathbf{0}$ if and only if the underlying shares of the $\zeta_1^i, \dots, \zeta_k^i$ are valid shares of a polynomial of degree $\leq \deg$. Finally, the parties compute

$$\mathbf{y}^j \leftarrow \sum_{l=1}^{d(m-\deg-1)/(4\varepsilon^2 n^2)} \beta^l \cdot \mathbf{y}_l^j,$$

then reconstruct it to the server who check if the reconstructed value is $\mathbf{0}$, and aborts if not. Otherwise, they abort.

Security Intuition Let t_{c_1} be the number of corrupted parties in committee \mathcal{C}_T that do not send to everyone in \mathcal{C}_{T+1} and $m = n - t_d - t_{c_1}$ be the number of parties from committee \mathcal{C}_T that do *not* drop out (including those corrupted parties that do not send). Writing $m = \deg + w + 1$, we have that $w = m - \deg - 1 = n - t_d - t_{c_1} - ((1/2 + \varepsilon)n - 1) - 1 = (1/2 - \varepsilon)n - t_d - t_{c_1} > t_{c_2}$, where t_{c_2} is the number of corrupted parties that do send to \mathcal{C}_{T+1} , and thus $t_{c_1} + t_{c_2} = t_c$. The last inequality holds, since we assume that $t_d + t_c < (1/2 - \varepsilon)n$. This means that if the corrupted parties from committee \mathcal{C}_T that do send, do not reshare their actual shares to committee \mathcal{C}_{T+1} , then the parity check sharing will not share $\mathbf{y}_i = \mathbf{0}$. This is because the number of honest parties who do not drop

⁶ $\text{negl}(\lambda)$ is any function in $\lambda^{\omega(1)}$

out is at least $\deg + 1$ and thus their shares completely define the correct polynomial and so if the corrupted parties' shares do not match with this polynomial, it will be reflected. Using similar logic, the server in round \mathcal{C}_{T+1} will be able to either successfully reconstruct the parity check sharing, or otherwise detect malicious behavior during the reconstruction.

Added Communication Complexity Note that most of the updates to achieve active security are done *locally*. The only added communication is for committing to and opening the randomness β_i , then reconstructing the y^i . Moreover, if we use the passively-secure protocol many times in parallel, then we can use the same β to take the random linear combination across all such instances. Thus the total communication complexity of the actively secure protocol is marginally changed with respect to the passively secure protocol, as long as if enough instances of the passive protocol are used at the same time.

Security Proof

Theorem 4 (Security). Π_{PPFL} securely computes $\mathcal{F}_{\text{PPFL}}$ with functionalities $\mathcal{F}_{\text{SecAgg}}$ and $\mathcal{F}_{\text{Comm}}$.

Proof. We first build the simulator \mathcal{S} . We first note that we model the SecAgg protocol as a trusted functionality $\mathcal{F}_{\text{SecAgg}}$ which takes inputs a_1, \dots, a_m from some parties via SecAgg.Enc and outputs their sum $\sum_{i=1}^m a_i$ to the server S via SecAgg.Dec. We also model commitments as a trusted functionality $\mathcal{F}_{\text{Comm}}$ that in the first stage takes in x from P_i and then does not reveal x to the other parties until the next stage. Indeed, the simulator emulates these trusted functionalities and thus can see whatever the corrupted parties input to them.

We describe the simulator for the first rounds $T = 1$ and then inductively for the rest. Throughout, we will (inductively) show that the simulator knows all of the corrupted parties' shares. We start with the case of a corrupted server S .

Corrupted Server In round 1, \mathcal{S} simulates the shares sent by honest parties of round 1 to corrupted parties of round 2 by sampling random values from the field \mathbb{F} . In round 2, \mathcal{S} receives on behalf of the honest parties in committee \mathcal{C}_2 the shares sent by corrupted parties from round 1. Note that the honest shares completely (and exactly) define these sharings since the number of honest parties is exactly $\deg + 1$, and thus \mathcal{S} can compute the corrupted parties' shares.

In subsequent rounds $T > 1$, \mathcal{S} first simulates the resharing of honest parties of round T to corrupted parties of round $T + 1$ by sampling random values from the field \mathbb{F} . In round $T + 1$, \mathcal{S} first inputs to $\mathcal{F}_{\text{Comm}}$ random β_i on behalf of the honest parties. It also receives on behalf of the honest parties in committee \mathcal{C}_{T+1} the reshared shares sent by corrupted parties from round T . Note that the honest shares completely (and exactly) define these sharings since the number of honest parties is exactly $\deg + 1$, and thus \mathcal{S} can compute the corrupted parties' shares as well as the actual underlying reshared shares $\hat{\zeta}_1^i, \dots, \hat{\zeta}_k^i$ of each corrupted party P_i in \mathcal{C}_T . Note that these might be different from the actual underlying shares $\zeta_1^i, \dots, \zeta_k^i$ of the corrupted parties which, inductively, \mathcal{S} knows. Thus, \mathcal{S} can compute $e_m^i \leftarrow \zeta_m^i - \hat{\zeta}_m^i$ for each $m \in [k]$. We have for $k \in [m]$:⁷

$$\mathbf{H} \cdot (\hat{\zeta}_m^1, \dots, \hat{\zeta}_m^n)^\top = \mathbf{H} \cdot (\zeta_m^1 + e_m^1, \dots, \zeta_m^1 + e_m^n)^\top = \mathbf{H}(e_m^1, \dots, e_m^n)^\top.$$

Since these are the underlying values of the shared vectors when the parties compute $\mathbf{H} \cdot (Z_1^j, \dots, Z_n^j)^\top$, \mathcal{S} can compute the underlying values of the shared vector defined by the shares y^j (also by using β). Thus, along with the corrupted parties' shares y^j , which it can compute manually with the corrupted parties' shares Z_m^j and β which it knows, it can reconstruct the honest parties' shares y^j and send these to the corrupted server.

Now we show that this is a good simulation. By the properties of Shamir Secret Sharing, we know that the at most t_c shares that the adversary receives in the real world for every sharing will be distributed randomly. Thus the shares that \mathcal{S} sends are distributed the same way. Also the y^j shares that \mathcal{S} sends to the corrupted server are computed exactly as they are in the real world, since \mathcal{S} can compute the e_m^i exactly and also inductively computes the corrupted parties' shares of all sharings exactly. Thus \mathcal{S} perfectly simulates the real world.

⁷For honest parties, $e_m^i = 0$.

679 **Honest Server** In the case of an honest server, we can use all of the same simulation above, except
680 we do not need to simulate the messages sent to the server. We do need to show that, even in the
681 presence of honest dropout parties, the random linear combinations of the parity checks do indeed
682 reconstruct to $\mathbf{0}$ if and only if the adversary did not tamper with its shares (which the simulator
683 can trivially check and abort if so, since it keeps track of the corrupted parties' shares). Since the
684 packed secret sharing scheme we use is linear, it is clear that applying the parity check matrix to the
685 shares of shares will result in shares of $\mathbf{0}$ if and only if the adversary reshared the correct underlying
686 shares: Let t_{c_1} be the number of corrupted parties in committee \mathcal{C}_T that do not send to everyone
687 in \mathcal{C}_{T+1} and $m = n - t_d - t_{c_1}$ be the number of parties from committee \mathcal{C}_T that do *not* drop
688 out (including those corrupted parties that do not send). Writing $m = \deg + w + 1$, we have that
689 $w = m - \deg - 1 = n - t_d - t_{c_1} - ((1/2 + \varepsilon)n - 1) - 1 = (1/2 - \varepsilon)n - t_d - t_{c_1} > t_{c_2}$, where t_{c_2}
690 is the number of corrupted parties that do send to \mathcal{C}_{T+1} , and thus $t_{c_1} + t_{c_2} = t_c$. The last inequality
691 holds, since we assume that $t_d + t_c < (1/2 - \varepsilon)n$. This means that if the corrupted parties from
692 committee \mathcal{C}_T that do send, do not reshare their actual shares to committee \mathcal{C}_{T+1} , then the parity
693 check sharing will not share $\mathbf{y}_i = \mathbf{0}$. This is because the number of honest parties who do not drop
694 out is at least $\deg + 1$ and thus their shares completely define the correct polynomial and so if the
695 corrupted parties' shares do not match with this polynomial, it will be reflected. Using similar logic,
696 the server in round \mathcal{C}_{T+1} will be able to either successfully reconstruct the parity check sharing, or
697 otherwise detect malicious behavior during the reconstruction.

698 In fact, this holds even after the parties take the random linear combination $\mathbf{y}^j \leftarrow$
699 $\sum_{l=1}^{d(n-\deg-1)/4\varepsilon^2n^2} \beta^l \cdot \mathbf{y}_l^j$, where d is the dimension of the model. This is because β was ran-
700 dom and unknown to the adversary before it generated its shares of shares. Thus, the underlying
701 values of this linear combination can be seen as the evaluation of a polynomial defined by coefficients
702 being the underlying values of the \mathbf{y}_l^j , on a random input β . By the Schwartz-Zippel Lemma, if any
703 of the underlying values of the $\mathbf{y}_l^j \neq \mathbf{0}$, then the result of this polynomial evaluation will not be $\mathbf{0}$
704 with probability $d(n - \deg - 1)/(4\varepsilon^2n^2 \cdot |\mathbb{F}|)$.⁸ Thus, if the adversary does not tamper with its shares
705 \mathbf{y}^j , then the reconstruction to the server will be $\mathbf{0}$ if and only if the adversary reshared the correct
706 shares. If the adversary does tamper with its shares \mathbf{y}^j , then we know by the properties of packed
707 secret sharing that the server will detect this and abort.

We also need to show that the output of the server is the same in the real and ideal worlds. Indeed, if
an adversary tampers with its shares before inputting them to SecAgg.Enc, the worst this can achieve
is an *additive attack* [24]: Let's consider the reconstruction of the shares of some $\widehat{\mathbf{A}\mathbf{X}}_T$ through
SecAgg, assuming w.l.o.g., that the first $d + 1$ parties are honest:

$$\sum_{i=1}^n \lambda_i^j \cdot \widehat{\mathbf{A}\mathbf{X}}_T^{i,tamp} = \sum_{i=1}^d \lambda_i^j \cdot \widehat{\mathbf{A}\mathbf{X}}_T^i + \sum_{i=d+1}^n \lambda_i^j \cdot (\widehat{\mathbf{A}\mathbf{X}}_T^i + \chi^i) = \widehat{\mathbf{A}\mathbf{X}}_T + \chi.$$

708 Indeed, since \mathcal{S} sees the values input to SecAgg.Enc by the corrupted parties and also inductively
709 knows what the corrupted parties' real input values should be, it can compute $\sum_{i=d+1}^n \lambda_i^j \cdot \chi^i$ and
710 thus χ . This completes the security proof. \square

711 C Discretization Details of [31]

712 We use the randomized rounding strategy from [31] for discretization in Π_{PPFL} . At a high-level, each
713 client first clips and scales their input gradient. Then, the clients flatten their gradient vectors using
714 some unitary matrix \mathbf{U} (intuitively, this minimizes the risk of modulo overlap in vector elements that
715 are particularly large). Finally, the clients use a randomized process to round their gradient vectors in
716 \mathbb{R}^d to \mathbb{Z}^d . On the server side, after receiving the aggregated, noise outputs $\widehat{\mathbf{A}\mathbf{X}}_T$ in each round, the
717 server unflattens the vector by applying \mathbf{U}^T and then descales. Protocols 2 and 3 give more detail,
718 but we refer the readers to [31] for full details on possible flattening matrices \mathbf{U} and the randomized
719 rounding procedure used.

⁸We assume that $|\mathbb{F}| > \lambda$.

Protocol 2 Client Gradient Processing

Input: Gradient $\mathbf{g}_i \in \mathbb{R}^d$.

Parameters: model dimension d , clipping threshold $c > 0$, granularity γ , modulus m , noise scale $\sigma > 0$ and bias $\beta \in [0, 1)$.

1. Clip and scale gradient: $\mathbf{g}'_i = \frac{1}{\gamma} \min\{1, \frac{c}{\|\mathbf{g}_i\|_2}\} \cdot \mathbf{g}_i \in \mathbb{R}^d$.
 2. Flatten vector: $\mathbf{g}''_i = U \cdot \mathbf{g}'_i \in \mathbb{R}^d$.
 3. **Repeat:**
 - (a) Let $\tilde{\mathbf{g}}_i \in \mathbb{Z}^d$ be a randomized rounding of \mathbf{g}''_i . i.e., $\tilde{\mathbf{g}}_i$ is a product distribution with $\mathbb{E}[\tilde{\mathbf{g}}_i] = \mathbf{g}''_i$ and $\|\tilde{\mathbf{g}}_i - \mathbf{g}''_i\|_\infty < 1$.
 - until** $\|\tilde{\mathbf{g}}_i\|_2 \leq \min\{c/\gamma + \sqrt{d}, \sqrt{c^2/\gamma^2 + \frac{1}{4}d} + \sqrt{2 \log(1/\beta)} \cdot (c/\gamma + \frac{1}{2}\sqrt{d})\}$.
 4. **Output:** $\tilde{\mathbf{g}}_i$.
-

Protocol 3 Server Aggregate Noisy Release Value Processing

Input: Vector \widehat{AX}_T .

Parameters: model dimension d , clipping threshold $c > 0$, granularity γ , modulus m , noise scale $\sigma > 0$ and bias $\beta \in [0, 1)$.

1. Map \mathbb{Z}_m to $\{1 - m/2, 2 - m/2, \dots, -1, 0, 1, \dots, m/2 - 1, m/2\}$ so that \widehat{AX}_T is mapped to $\widehat{AX}'_T \in [-m/2, m/2]^d \cap \mathbb{Z}^d$ (and we have $\widehat{AX}'_T \bmod m = \widehat{AX}_T$).

Output: $\gamma \cdot U^\top \widehat{AX}'_T \in \mathbb{R}^d$.

D Additional Experimental Results

Here we empirically evaluate our Distributed Matrix Mechanism (DMM) for Federated Learning on the Stack Overflow Next Word Prediction public benchmark [4], as in [31, 15]. Stack Overflow is a large-scale text dataset based on the question answering site Stack Overflow. It contains over 108 training sentences extracted from the site grouped by the $N = 342477$ users, and each sentence has associated metadata such as tags. The task of SO-NWP involves predicting the next words given the preceding words in a sentence. We use the standard dataset split provided by TensorFlow. We compare to the Distributed Discrete Gaussian Mechanism for FL [31] that also obtains local DP, but with independent noise and reliance upon privacy amplification via sampling [1, 33, 7], as well as the central DP version of our paper for multiple epochs [15], where noise is correlated, but the server applies it.

As in [31, 15], we use the LSTM architecture defined in [42] directly, which has a model size of $d = 4050748$ parameters (slightly under 2^{22}). We use namely momentum 0.9, 1 client training epoch per round, client learning rate $\eta_c = 0.02$, server learning rate $\eta_s = 1$, and client batch size to 16. For Π_{PPFL} , we assume that $\mu = 1/6$; i.e., the number of corrupted parties and dropout parties per round satisfies $t_c + t_d < 1/3n$.

Matrix Factorizations As for EMNIST, we use two different matrix factorizations $\mathbf{A} = \mathbf{BC}$ for our experiments. The first is the optimal with respect to the loss function $\mathcal{L}(\mathbf{B}, \mathbf{C}) = \text{sens}_\Phi(\mathbf{C}) \|\mathbf{B}\|_F^2$ for the (ν, b) -participation schema Φ , as generated by the code from [15]. The second is the Honaker Online mechanism [32, 28], where \mathbf{C} is essentially the binary tree matrix. Again, this mechanism has the benefit that it allows for implementations with only $\log(T^*)$ overhead; i.e., in the T -th round, the released model can be computed using at most $d \cdot \log(T^*)$ values. Thus, the

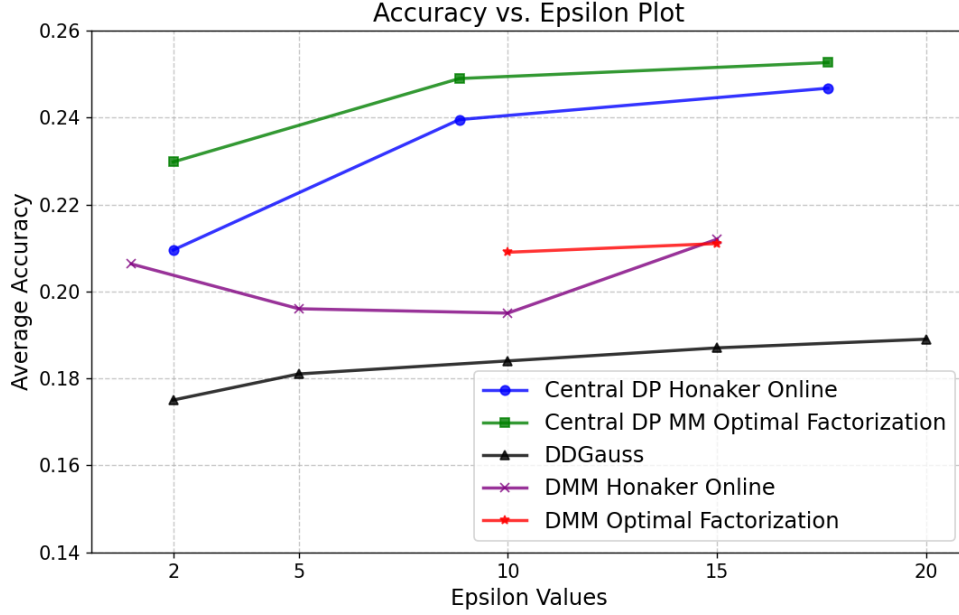


Figure 4: Test accuracies on SO NWP across different ε for the DDG mechanism [31], the central-DP matrix mechanism for multiple epochs [15], and our DMM instantiated with the optimal factorization for multiple epochs and the Honaker online factorization.

size of the secret vectors that must be reshared from one committee to the next are at most $d \cdot \log(T^*)$ instead of $d \cdot T^*$, which greatly increases efficiency, as we will see below.

Results Figure 4 shows that for several different ε privacy levels, our DMM significantly outperforms the DDGauss Mechanism in terms of prediction accuracy, while getting close to that of the central-DP matrix mechanism of [15]. We also see that the Honaker mechanism only sees slight accuracy degradation compared to the mechanism based on the optimal (ν, b) -participation matrix factorization. Therefore, the tree mechanism might be best in practice due to much better efficiency. These experiments all use $n = 40$ clients per round. For the tree mechanism, we use $T^* = 2^{10} = 1024$ and for the optimal matrix factorization, we use $T^* = 1500$; this corresponds to $\nu = 13, b = 85$ and $\nu = 18, b = 85$, respectively.

E Attacks on Other Approaches and Future Work

Instead of maintaining secret-shared versions of the aggregated gradients and noise vectors, the server could preserve the aggregated noise vectors and gradients of previous training iterations within the system by masking them with an appropriate mask mk invoking a secure aggregation protocol SecAgg_1 . The masks mk themselves would be secret shared and reshared among the clients. That said, the black-box secure aggregation SecAgg_1 protocol would output aggregated gradients G and noise vectors masked by mk , i.e., $G + mk$ to the server. When it is time to aggregate in each training iteration, another black-box SecAgg_2 protocol is called in which the server would input the masked aggregated gradients and noise vectors along and the clients would input the negative shares of the masks mk . This ensures that the secure aggregation SecAgg_2 protocol outputs the unmasked (the masks of the gradients and noise vectors from previous iterations would cancel out) noisy aggregate for the current iteration to the server.

However, this approach faces a fundamental issue: the server holds the masked aggregated noise and gradients and could input any dishonest combination into the aggregation protocol to undermine DP. Specifically, the server might:

- **Selective Noise Cancellation:** In the matrix mechanism, noise is added directly by the clients in the current training iteration, and past aggregated correlated noise is added to enhance

769 utility by canceling out some of the total noise. If the server has access to the masked
770 aggregated noise, it could selectively include or exclude certain masked noises as input
771 to the secure aggregation protocol SecAgg_2 , effectively canceling out noise terms across
772 training iterations. This would enable selective noisy cancellation, potentially weakening
773 the overall differential privacy guarantees.

774 • **Manipulation of Scaled Aggregated Gradients:** The server might multiply the aggregated
775 masked gradients by a malleable value when inputting them into the secure aggregation
776 protocol SecAgg_2 , causing the noise to be incorrectly scaled relative to the proper sensitivity.
777 This manipulation could reveal information about the current iteration’s aggregated gradients,
778 thereby compromising the privacy guarantees.

779 **Future work** An alternative method for rolling noise forward to the next committee is to encrypt
780 the noise rather than secret-sharing it based on our resharing protocol. However, an efficient solution
781 is not straightforward, as the noise must remain encrypted while being used by the clients. The
782 challenge lies in determining which keys to use for encryption. If the noise is encrypted using the
783 server’s key, the server could decrypt it, compromising privacy. Conversely, if it is encrypted under
784 the client’s keys, they would be able to decrypt it. Identifying an advanced encryption scheme that can
785 maintain privacy and offer better efficiency remains an open question for future research.

786 **F Π_{PPFL} without the Use of SecAgg**

787 In this section, we present a version of Π_{PPFL} (below) without relying on any black-box secure
788 aggregation protocol. Specifically, since the clients secret share both the noise vectors and the
789 gradients, we perform secure aggregation in Π_{PPFL} by having all clients within the same committee
790 in the same training iteration receive these shares (via the server), aggregate them, and send the result
791 back to the server, which recovers the final noisy sum. This requires an extra communication round
792 per training iteration. However, it’s worth noting that current secure aggregation SecAgg protocols
793 already require at minimum two rounds of interaction.

794 The advantage of presenting Π_{PPFL} in Section 5 with access to a secure aggregation SecAgg pro-
795 tocol in a black-box manner is that future advancements might lead to a significantly faster secure
796 aggregation protocol. As it stands, our protocol, based solely on secret sharing and our resharing
797 protocol, efficiently packs multiple secrets into a single share, resulting to a communication overhead
798 that asymptotically matches existing secure aggregation SecAgg protocols.

Protocol 4 Privacy-Preserving Federated Learning Protocol Π_{PPFL} without black box us of SecAgg

Protocol Π_{PPFL} runs with clients P_1, \dots, P_N and a server S . Let $\text{PSS} = (\text{Share}, \text{Reshare}, \text{Reconstruct}, \text{Recover})$ be a packed resharing protocol (See Section 3). $\Pi_{\text{PPFL}} = (\text{Setup}, \text{Agg})$ proceeds as follows:

Parameters: Dimension $d \in \mathbb{N}$; clipping threshold $c > 0$; granularity $\gamma > 0$; noise scale $\sigma > 0$; bias $\beta \in [0, 1]$; finite field \mathbb{F} of bit-width m ; public (lower-triangular) matrix encoding of prefix sums or stochastic gradient descent with momentum (SGDM) [17]) $A \in \mathbb{R}^{T^* \times T^*}$; matrices B, C such that $A = BC$.

Inputs: For $i \in [N]$, party P_i holds input dataset D_i . Without loss of generality we assume that committees in each training iteration are of size n .

Agg($D_i, Y_{T-1}, G_{T-1}, \zeta_{T-1}, \{X_\tau, Z_\tau\}_{\tau \in [T-2]}$): Let \mathcal{C}_T be the set of chosen clients for the T -th training iteration. For each T each client P_i in \mathcal{C}_T proceeds as follows:

Round 1:

- Runs training model on Y_{T-1}, D_i which generates the vector of local gradients g_i (that are then clipped to norm c , scaled via granularity parameter $\gamma > 0$, flattened, and rounded/discretized with bias $\beta \in [0, 1]$ as in [31]; details of this are provided in Section C).
- Samples a noise vector z_i from a Discrete Gaussian distribution $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2)$.
- Secret shares the noise vector and the gradients using the packed secret sharing scheme as $\zeta_T^i = \text{Share}(z_i)$ and $G_T^i = \text{Share}(g_i)$ to the set \mathcal{C}_T . Each j -th share of ζ_T^i and X_T^i is encrypted to the j -th client of \mathcal{C}_T using authenticated and encrypted channels (via the Server).

Round 2:

- Decrypts and aggregates the shares (received by the server) of the noise vector and gradients $Z_T^i = (\sum_j \zeta_T^{j,i})$ and $X_T^i = (\sum_j G_T^{j,i})$ and securely reshapes them using the packed resharing protocol as $Z_{T+1}^i = \text{Reshare}(Z_T^i)$, $X_{T+1}^i = \text{Reshare}(X_T^i)$ to the set of clients in \mathcal{C}_{T+1} for the next training iteration.
- For the j -th model parameter in each batch of parameters sends to S :

$$y_i = \text{Reconstruct} \left(i, \sum_{\tau=1}^T (A_{[T,\tau]} \cdot X_\tau^i + B_{[T,\tau]} \cdot Z_\tau^i), j \right).$$

If $T > 1$:

- Decrypts and reshapes all the previous τ aggregated shares as $X_T^i = \text{Reshare}(X_\tau^i)$ and $Z_T^i = \text{Reshare}(Z_\tau^i)$ to set \mathcal{C}_{T+1} for $\tau \in [1, T-1]$.

Round 3:

1. S recovers the noisy sums as $Y_T = (\sum_i y_i)$.
-