APPENDIX

A CLARIFICATION ON PRIVACY

As indicated in Sec. 2, the projection matrix is public information, and hence FED-χ^2 does not take the differential privacy guarantee into account. We would want to provide more information in order to eliminate any potential misunderstandings.

To begin, we would want to emphasize that "privacy" in our paper refers to MPC-style privacy, not DP-style privacy. In general, MPC-style privacy is orthogonal to DP-style privacy: in MPC, privacy is obtained against a semi-honest server in such a way that the server cannot witness individual client's updates but only an aggregate of them, e.g. SecAgg (Bonawitz et al., 2017). In DP, privacy is accomplished by including random noise in each client's update, such that the distribution of the output result does not reveal the clients' private information and the server cannot infer the clients' identification from the output result.

Second, we would like to emphasize that our work proposes a novel secure aggregation scheme particularly for the χ^2-test. Existing standard secure aggregation schemes are inapplicable to the χ^2-test, which will reveal much more information than FED-χ^2, as we have clarified in Sec. 1. Again, this work requires guaranteeing MPC-style privacy, not DP.

Third, to quantify MPC-style privacy, we prove in Theorem 2 that the clients' updates in FED-χ^2 are hidden inside a space with exponential size. This is weaker than hiding users' updates in the whole space, but still gives meaningful privacy guarantees (consider attempting to guess the output of an exponential-sided dice, which is practically infeasible).

Finally, while DP is orthogonal to this research, we would want to emphasize that our protocol can achieve DP by introducing calibrated discrete Gaussian noise to the users' local updates.

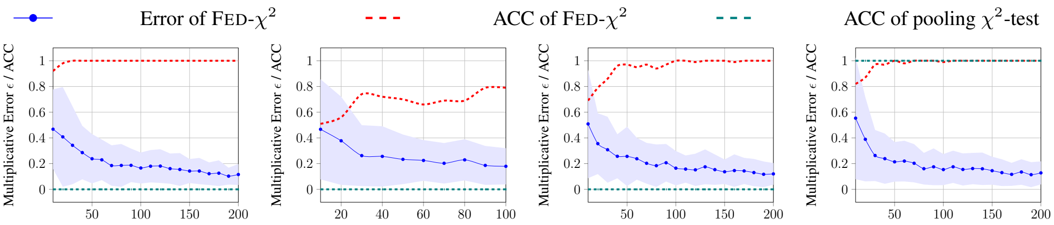
B COMPARISON WITH POOLING χ^2-TEST

We also compare the performance of FED-χ^2 with pooling χ^2-test. That is, the clients compute the χ^2-test with their local observations and then they aggregate their test results by pooling. The result of the pooling χ^2-test is determined by the majority of the clients' local results. Fig. 6 shows the result of pooling χ^2-test on the real-world datasets. The details of the datasets are presented in Appendix I. We observe that pooling χ^2-test cannot give meaningful results and it tends to give judgement that the data is independent since that the numbers of the local observations are not sufficient to make correct judgement. The results further demonstrate the effectiveness and the necessity of FED-χ^2.

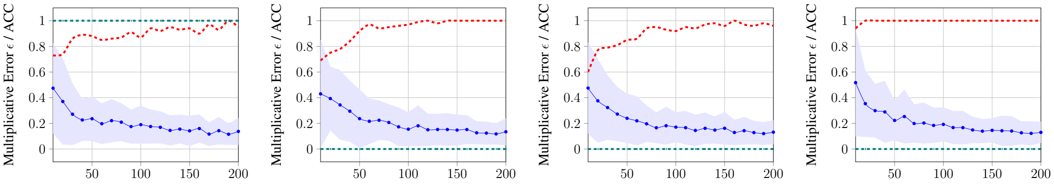
C PERFORMANCE OF FED-χ^2 WHEN ORIGINAL χ^2-TEST ACHIEVES LOW POWER

We have computed the multiplicative error and accuracy of FED-χ^2 with the original centralized χ^2-test as the baseline. To further demonstrate the effectiveness of FED-χ^2, we also set up the experiment to evaluate the performance of FED-χ^2 on the synthesized dataset when the accuracy of the original centralized χ^2-test is lower. More specifically, we synthesize the random independent, linearly correlated, quadratically correlated, and logistically correlated datasets with the same hyper-parameters for 20 times. We then compute the accuracy of the original centralized χ^2-test over these 20 datasets. We report the results in Fig. 7. The results show that FED-χ^2 can still achieve good performance on the datasets when the original centralized χ^2-test's accuracy is lower. Consistent with our results in Sec. 4.1, the multiplicative error becomes lower with the increase of the encoding size l. Thus, we conclude that FED-χ^2's performance is comparable to that of the original centralized χ^2-test with an appropriate encoding size l and is not dependent on the datasets.

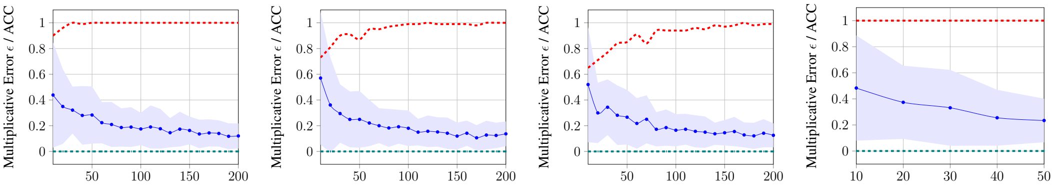
D FURTHER RESULTS FOR ONLINE FDR CONTROL



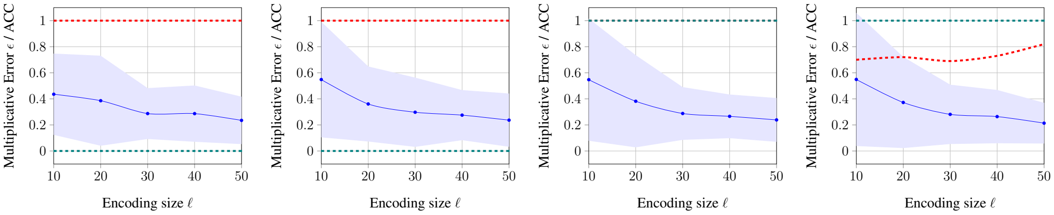
(a) Data 1. (b) Data 2. (c) Data 3. (d) Data 4.



(e) Data 5. (f) Data 6. (g) Data 7. (h) Data 8.

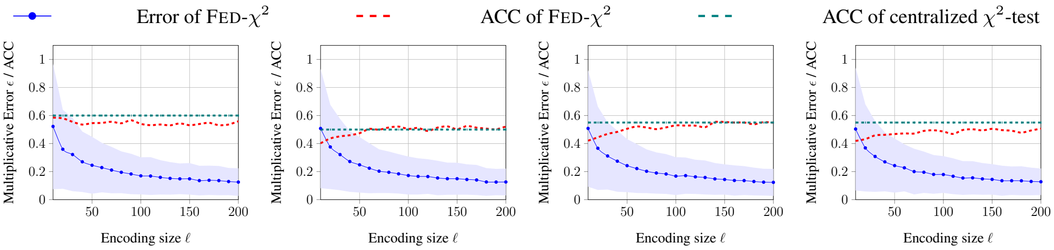


(i) Data 9. (j) Data 10. (k) Data 11. (l) Data 12.



(m) Data 13. (n) Data 14. (o) Data 15. (p) Data 16.

Figure 6: Comparison between FED-χ^2 and pooling χ^2-test.



(a) Synthetic Data 1. (b) Synthetic Data 2. (c) Synthetic Data 3. (d) Synthetic Data 4.

Figure 7: Performance of FED-χ^2 when original χ^2-test achieves low accuracy.

In this section, we provide further results for online FDR control. As we have shown in Fig. 5, FED-χ^2 achieves good performance (FDR lower than 5%) when the encoding size l is larger than 200. In Fig. 8, we provide the FDR result of the original χ^2-test as well as the true discovery rate (TDR, i.e., #correct reject / #should reject). In addition, we provide statistics for each encoding size l that was evaluated in Table 1. These results demonstrate that FED-χ^2 performs well and is comparable to the centralized χ^2-test when the encoding size l is increased. More importantly, as we have shown in Sec. 4.1 and Appendix C, the performance of FED-χ^2 is independent from the data.

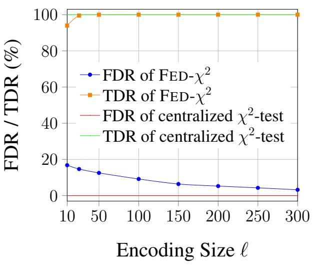


Figure 8: Results of FDR & TDR.

	#should reject	#should accept	#correct reject	#false reject
FED-χ^2, $l = 10$	5,900	4,100	5,544	1,132
FED-χ^2, $l = 25$	5,900	4,100	5,871	1,022
FED-χ^2, $l = 50$	5,900	4,100	5,899	856
FED-χ^2, $l = 100$	5,900	4,100	5,900	606
FED-χ^2, $l = 150$	5,900	4,100	5,900	411
FED-χ^2, $l = 200$	5,900	4,100	5,900	335
FED-χ^2, $l = 250$	5,900	4,100	5,900	270
FED-χ^2, $l = 300$	5,900	4,100	5,900	202
centralized χ^2-test	5,900	4,100	5,900	0

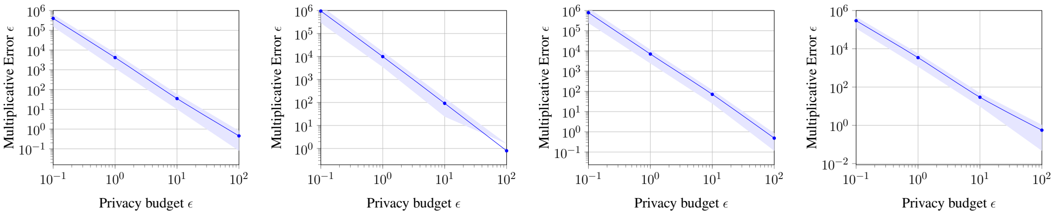
Table 1: Detailed results of online FDR control.

E INCORPORATE GAUSSIAN MECHANISM IN FED-χ^2

As we have mentioned in Appendix A, FED-χ^2 can achieve differential privacy easily by incorporating well-studied differentially private mechanisms. To further demonstrate this point, we utilize Gaussian Mechanism to provide (ϵ, δ)-DP guarantee. We clipped the local clients' data such that the encoding function's sensitivity, which is the L2-norm of the clients' local data is bounded by Δf. Before encoding their local data, each client add Gaussian noise $\mathcal{N}^d(0, \sigma^2)$ to their local data vector μ_i. We can calculate $\sigma^2 = \frac{2\ln(1.25/\delta)(\Delta f)^2}{n(\epsilon/l)^2}$. After encoding and decoding with the computed vector, FED-χ^2 provides $(\epsilon, \delta)-$DP guarantee.

We evaluate the performance of differentially private FED-χ^2 on four of the real-world datasets as shown in Fig. 9. The results show that the performance our algorithm becomes better with the increase of privacy budget ϵ. When the privacy budget ϵ is small, the protocol tends to give the judgement that the data is independent because that the independent noise is too large and it dominates the test. When the privacy budget ϵ is large enough, our protocol can achieve 0.92, 0.68, 0.70, and 0.80 accuracy on these datasets accordingly, and its performance is comparable to the original FED-χ^2.

Again, our work is orthogonal to differential privacy, thus we leave it as future work to further study saving privacy budget, and boosting the algorithm's performance on lower ϵ.



(a) Data 1.

(b) Data 2.

(c) Data 3.

(d) Data 4.

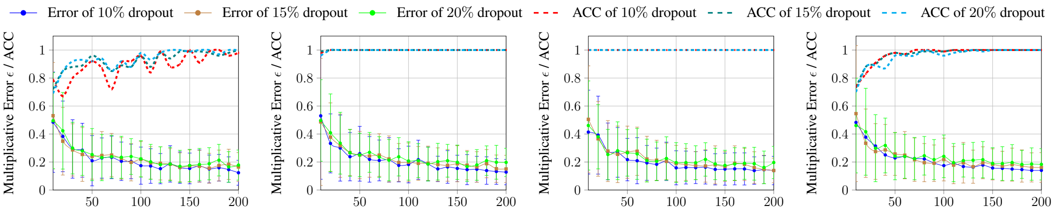
Figure 9: Performance of FED-χ^2 when incorporate with Gaussian Mechanism.

F FURTHER RESULTS ON FED-χ^2 WITH DROPOUTS

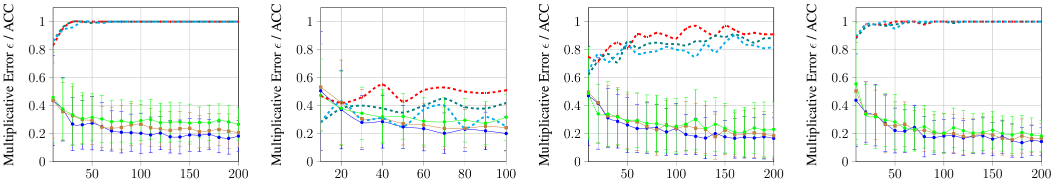
We present the results of 10%, 15%, and 20% clients dropout in Fig. 10. The results further show that FED-χ^2 can tolerate a considerable portion of clients dropout in Round 2 of Alg. 2.

G SECURE AGGREGATION

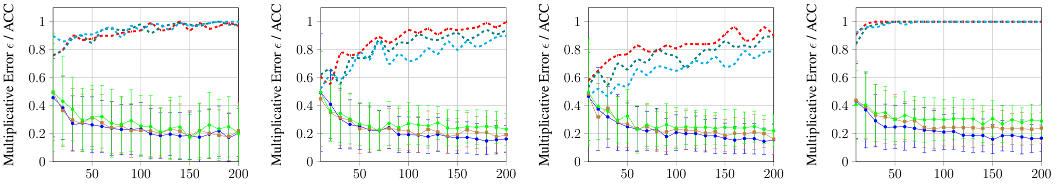
The secure aggregation protocol from Bell et al. (2020) is presented in Alg. 4. The first step of the protocol is to generate a k-regular graph G, where the n vertices are the clients participating in the protocol. The server runs a randomized graph generation algorithm INITSECUREAGG presented in Alg. 3 that takes the number of clients n and samples output (G, t, k) from a distribution $\mathcal{D}$. In Alg. 3, we uniformly rename the nodes of a graph known as a Harary graph defined in Definition 3 with n nodes and k degrees. The graph G is constructed by sampling k neighbours uniformly and without replacement from the set of remaining $n-1$ clients. We choose $k = \mathcal{O}(log(n))$, which is large enough to hide the updates inside the masks. t is the threshold of the Shamir's Secret Sharing.



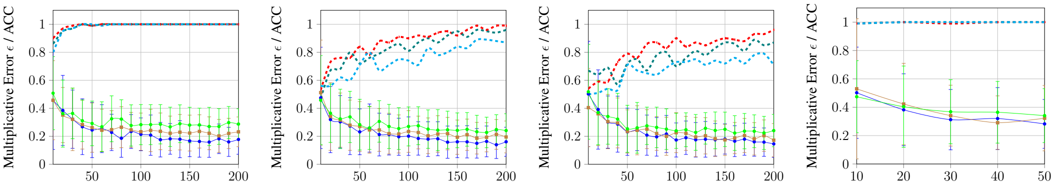
(a) Synthetic Data 1. (b) Synthetic Data 2. (c) Synthetic Data 3. (d) Synthetic Data 4.



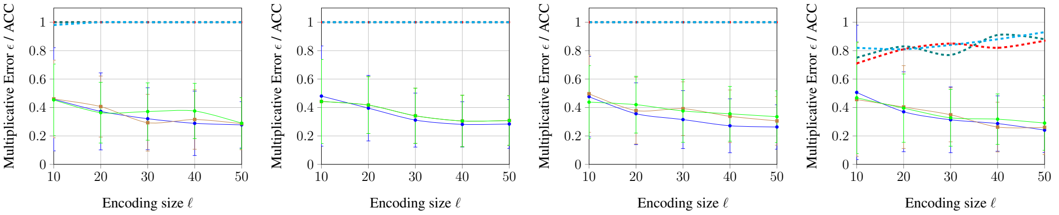
(e) Data 1. (f) Data 2. (g) Data 3. (h) Data 4.



(i) Data 5. (j) Data 6. (k) Data 7. (l) Data 8.



(m) Data 9. (n) Data 10. (o) Data 11. (p) Data 12.



(q) Data 13. (r) Data 14. (s) Data 15. (t) Data 16.

Figure 10: Multiplicative error and accuracy of FED-χ^2 w.r.t. encoding size ℓ w/ and w/o dropout.

In the second step, the edges of the graph determine pairs of clients, each of which runs key agreement protocols to share random keys. The random keys will be used by each party to derive a mask for her input and enable dropouts.

In the third step, each client $c_i, i \in A_1$ sends secret share to its neighbors. In the fourth step, the server checks whether the clients dropout exceeds the threshold δ, and lets the clients know their neighbors who didn't dropout.

In the fifth step, each pair (i, j) of connected clients in G runs a λ-secure key agreement protocol $s_{i,j} = \mathcal{KA}.Agree(sk_i^1, pk_j^1)$ which uses the key exchange in the previous step to derive a shared random key $s_{i,j}$. The pairwise masks $\mathbf{m}_{i,j} = F(s_{i,j})$ can be computed, where F is the pseudorandom generator (PRG). If the semi-honest server announces dropouts and later some masked inputs of the claimed dropouts arrive, the server can recover the inputs. To prevent this happening, another level of masks, called self masks, $\mathbf{r}_i$ is added to the input. Thus, the input of client c_i is: $\mathbf{y}_i = \mathbf{e}_i + \mathbf{r}_i - \sum_{j \in N_G(i), j<i} \mathbf{m}_{i,j} + \sum_{j \in N_G(i), j>i} \mathbf{m}_{i,j}$.

Steps 6–8 deal with the clients dropout by recovering the self masks $\mathbf{r}_i$ of clients who are still active and pairwise masks $\mathbf{m}_{i,j}$ of the clients who have dropped out. Finally, the server can cancel out the pairwise masks and subtract the self masks in the final sum: $\sum_{i\in A'_2}(\mathbf{y}_i - \mathbf{r}_i + \sum_{j\in NG(i)\cap(A'_1\setminus A'_2),0<j<i}\mathbf{m}_{i,j} - \sum_{j\in NG(i)\cap(A'_1\setminus A'_2),i<j\leq n}\mathbf{m}_{i,j})$.

Definition 3 (HARARY(n,k) Graph)**.** *Let* HARARY(n,k) *denotes a graph with* n *nodes and degree* k*. This graph has vertices* $V=[n]$ *and an edge between two distinct vertices* i *and* j *if and only if* $j-i \pmod n \leq (k+1)/2$ *or* $j-i \pmod n \geq n-k/2$.

Algorithm 3: INITSECUREAGG: Generate Initial Graph for SECUREAGG.

```
Function InitSecureAgg($n$):
    ▷ $n$: Number of nodes.
    ▷ $t$: Threshold of Shamir's Secret Sharing.
    $k = \mathcal{O}(log(n))$.
    Let $H$ = Harary($n, k$).
    Sample a random permutation $\pi : [n] \to [n]$.
    Let $G$ be the set of edges $\{(\pi(i), \pi(j)) | (i,j) \in H\}$.
    return $(G, t, k)$
```

H PROOF FOR COMMUNICATION & COMPUTATION COST

We provide the proof for Theorem 4 and Theorem 5 in the following.

Theorem 4 (Communication Cost). Let Π be an instantiation of Alg. 2 with secure aggregation protocol from Bell et al. (2020), then (1) the client-side communication cost is $\mathcal{O}(\log n + m_x + m_y + \ell)$; (2) the server-side communication cost $\mathcal{O}(n\log n + nm_x + nm_y + n\ell)$.

Proof sketch for Theorem 4. Each client performs k key agreements ($\mathcal{O}(k)$ messages, line 9 in Alg. 4) and sends 3 masked inputs ($\mathcal{O}(m_x + m_y + \ell)$ complexity, lines 3, 4, 15 in Alg. 2 and line 10 in Alg. 4). Thus, the client communication cost is $\mathcal{O}(\log n + m_x + m_y + \ell)$.

The server receives or sends $\mathcal{O}(\log n + m_x + m_y + \ell)$ messages to each client, so the server communication cost is $\mathcal{O}(n\log n + nm_x + nm_y + n\ell)$. ☐

Theorem 5 (Computation Cost). Let Π be an instantiation of Alg. 2 with secure aggregation protocol from Bell et al. (2020), then (1) the client-side computation cost is $\mathcal{O}(m_x \log n + m_y \log n + \ell \log n + m\ell)$; (2) the server-side computation cost is $\mathcal{O}(m_x + m_y + \ell)$.

Proof sketch for Theorem 5. Each client computation can be broken up as k key agreements ($\mathcal{O}(k)$ complexity, line 9 in Alg. 4), generating masks $\mathbf{m}_{i,j}$ for all neighbors c_j ($\mathcal{O}(k(m_x + m_y + \ell))$ complexity, lines 3, 4, 15 in Alg. 2 and line 10 in Alg. 4), and encoding computation cost $\mathcal{O}(m\ell)$ (line 14 in Alg. 2). Thus, the client computation cost is $\mathcal{O}(m_x \log n + m_y \log n + \ell \log n + m\ell)$.

The server-side follows directly from the semi-honest computation analysis in Bell et al. (2020). The extra $\mathcal{O}(\ell)$ term is the complexity of the geometric mean estimator. ☐

I DETAILS OF DATASETS

The details for the real-world datasets used in Sec. 4.1 are provided in Table 2. The license of Credit Risk Classification (Govindaraj, Praveen) is CC BY-SA 4.0, the license of German Traffic Sign (Houben et al., 2013) is CC0: Public Domain. Other datasets without a license are from UCI Machine Learning Repository (Dua & Graff, 2017).

J DETAILS OF REGRESSION MODELS

The details of the regression models trained in feature selection in Sec. 4.2 is reported in Table 3. The training and testing splits are the same for FED-χ^2, centralized χ^2-test and model without feature

Algorithm 4: SECUREAGG: Secure Aggregation Protocol. (Algorithm 2 from Bell et al. (2020))

```
Function SecureAgg ($\{e_i\}_{i\in[n]}$):
    ▷ Parties: Clients $c_1, \cdots, c_n$, and Server.
    ▷ $l$: Vector length.
    ▷ $\mathbb{X}^l$: Input domain, $\mathbf{e}_i \in \mathbb{X}^l$.
    ▷ $F : \{0,1\}^\lambda \to \mathbb{X}^l$: PRG.
    ▷ We denote by $A_1, A_2, A_3$ the sets of clients that reach certain points without dropping out.
      Specifically $A_1$ consists of the clients who finish step (3), $A_2$ those who finish step (5), and
      $A_3$ those who finish step (7). For each $A_i$, $A_i'$ is the set of clients for which the server sees
      they have completed that step on time.
    (1) The server runs $(G, t, k) =$ InitSecureAgg ($n$), where $G$ is a regular degree-$k$
      undirected graph with $n$ nodes. By $N_G(i)$ we denote the set of $k$ nodes adjacent to $c_i$ (its
      neighbors).
    (2) Client $c_i, i \in [n]$, generates key pairs $(sk_i^1, pk_i^1)$, $(sk_i^2, pk_i^2)$ and sends $(pk_i^1, pk_i^2)$ to the
      server who forwards the message to $N_G(i)$.
    (3) for each Client $c_i, i \in A_1$ do
        • Generates a random PRG seed $b_i$.
        • Computes two sets of shares:
              $H_i^b = \{h_{i,1}^b, \cdots, h_{i,k}^b\} = ShamirSS(t, k, b_i)$
              $H_i^s = \{h_{i,1}^s, \cdots, h_{i,k}^s\} = ShamirSS(t, k, sk_i^1)$
        • Sends to the server a message $m = (j, c_{i,j})$, where $c_{i,j} = \mathcal{E}_{auth}.Enc(k_{i,j}, (i||j||h_{i,j}^b||h_{i,j}^s))$ and
          $k_{i,j} = \mathcal{KA}.Agree(sk_i^2, pk_j^2)$, for each $j \in N_G(i)$.
    (4) The server aborts if $|A_1'| < (1-\delta)n$ and otherwise forwards $(j, c_{i,j})$ to client $c_j$ who
      deduces $A_1' \cap N_G(j)$.
    (5) for each Client $c_i, i \in A_2$ do
        • Computes a shared random PRG seed $s_{i,j}$ as $s_{i,j} = \mathcal{KA}.Agree(sk_i^1, pk_j^1)$.
        • Computes masks $\mathbf{m}_{i,j} = F(s_{i,j})$ and $\mathbf{r}_i = F(b_i)$.
        • Sends to the server their masked input
              $\mathbf{y}_i = \mathbf{e}_i + \mathbf{r}_i - \sum_{j\in[n],j<i} \mathbf{m}_{i,j} + \sum_{j\in[n],j>i} \mathbf{m}_{i,j}$
    (6) The server collects masked inputs. It aborts if $|A_2'| < (1-\delta)n$ and otherwise sends
      $(A_2' \cup N_G(i), (A_1 \backslash A_2') \cup N_G(i))$ to every client $c_i, i \in A_2'$.
    (7) Client $c_j, j \in A_3$ receives $(R_1, R_2)$ from the server and sends
      $\{(i, h_{i,j}^b)\}_{i\in R_1} \cup \{(i, h_{i,j}^s)\}_{i\in R_2}$ obtained by decrypting the $c_{i,j}$ received in Step (3).
    (8) The server aborts if $|A_3'| < (1-\delta)n$ and otherwise:
        • Collects, for each client $c_i, i \in A_2'$, the set $B_i$ of all shares in $H_i^b$ sent by clients in $A_3$. Then
          aborts if $|B_i| < t$ and otherwise recovers $b_i$ and $\mathbf{r}_i$ using the $t$ shares received which came from
          the lowest client IDs.
        • Collects, for each client $c_i, i \in (A_1 \backslash A_2')$, the set $S_i$ of all shares in $H_i^s$ sent by clients in $A_3$.
          Then aborts if $|S_i| < t$ and otherwise recovers $sk_i^1$ and $\mathbf{m}_{i,j}$.
        • return $\sum_{i\in A_2'}(\mathbf{y}_i - \mathbf{r}_i + \sum_{j\in NG(i)\cap(A_1'\backslash A_2'),0<j<i} \mathbf{m}_{i,j} - \sum_{j\in NG(i)\cap(A_1'\backslash A_2'),i<j\le n} \mathbf{m}_{i,j})$.
```

selection (i.e. there are 17,262 training and 4,316 test documents). We use the same learning rate; random seed and all other settings are also the same to make the comparison fair. We get the result of Fig. 3 and the models are all trained on NVIDIA GeForce RTX 3090.

Table 2: Dataset details.

ID	Data	Attr #1	A#1 Cat	Attr #2	A#2 Cat
1	Adult Income (Kohavi, 1996; Kohavi, Ronny and Becker, Barry)	Occupation	14	Native Country	41
2	Credit Risk Classification (Govindaraj, Praveen)	Feature 6	14	Feature 7	11
3	Credit Risk Classification (Govindaraj, Praveen)	Credit Product Type	28	Overdue Type I	35
4	Credit Risk Classification (Govindaraj, Praveen)	Credit Product Type	28	Overdue Type II	35
5	Credit Risk Classification (Govindaraj, Praveen)	Credit Product Type	28	Overdue Type III	36
6	German Traffic Sign (Houben et al., 2013)	Image Width	219	Traffic Sign	43
7	German Traffic Sign (Houben et al., 2013)	Image Height	201	Traffic Sign	43
8	German Traffic Sign (Houben et al., 2013)	Upper left X coordinate	21	Traffic Sign	43
9	German Traffic Sign (Houben et al., 2013)	Upper left Y coordinate	16	Traffic Sign	43
10	German Traffic Sign (Houben et al., 2013)	Lower right X coordinate	204	Traffic Sign	43
11	German Traffic Sign (Houben et al., 2013)	Lower right Y coordinate	186	Traffic Sign	43
12	Mushroom (Schlimmer, Jeff)	Cap color	10	Odor	9
13	Mushroom (Schlimmer, Jeff)	Gill color	12	Stalk color above ring	9
14	Mushroom (Schlimmer, Jeff)	Stalk color below ring	9	Ring Type	8
15	Mushroom (Schlimmer, Jeff)	Spore print color	9	Habitat	7
16	Lymphography (Kononenko, Igor and Cestnik, Bojan)	Structure Change	8	No. of nodes	8

Table 3: Model details.

Task	Model Size	Learning Rate	Random Seed
FED-χ^2	40000×20	0.1	0
Centralized χ^2-test	40000×20	0.1	0
Without Feature Selection	167135×20	0.1	0

K SAFFRON PROCEDURE

In Sec. 4.2, we adopt the SAFFRON procedure (Ramdas et al., 2018) to perform online FDR control. SAFFRON procedure is currently the state of the arts for multiple hypothesis testing. In Alg. 5, we formally present the SAFFRON algorithm.

Algorithm 5: SAFFRON Procedure.

```
1  Function SAFFRONPROCEDURE($\{p_1, p_2, \cdots\}$, $\alpha$, $W_0$, $\{\gamma_j\}_{j=0}^{\infty}$):
2      ▷ $\{p_1, p_2, \cdots\}$: Stream of $p$-values.
3      ▷ $\alpha$: Target FDR level.
4      ▷ $W_0$: Initial wealth.
5      ▷ $\{\gamma_j\}_{j=0}^{\infty}$: Positive non-increasing sequence summing to one.
6      $i \leftarrow 0$                                  // Set rejection number.
7      for each p-value $p_t \in \{p_1, p_2, \cdots\}$ do
8          $\lambda_t \leftarrow g_t(R_{1:t-1}, C_{1:t-1})$
9          $C_t \leftarrow I(p_t < \lambda_t)$            // Set the indicator for candidacy $C_t$.
10         $C_{j+} \leftarrow \sum_{i=\tau_j+1}^{t-1} C_i$      // Set the candidates after the $j^{th}$
            rejection.
11         if $t = 1$ then
12             $\alpha_1 \leftarrow (1 - \lambda_1)\gamma_1 W_0$
13         else
14             $\alpha_t \leftarrow (1 - \lambda_t)(W_0\gamma_{t-C_{0+}} + (\alpha - W_0)\gamma_{t-\tau_1-C_{1+}} + \sum_{j\geq 2} \alpha\gamma_{t-\tau_j-C_{j+}})$
15         $R_t \leftarrow I(p_t \leq \alpha_t)$                          // Output $R_t$.
16         if $R_t = 1$ then
17             $i \leftarrow i + 1$                       // Update rejection number.
18             $\tau_i \leftarrow t$                       // Set the $i^{th}$ rejection time.
19     return $\{R_0, R_1, \cdots\}$
```

The initial error budget for SAFFRON is $(1 - \lambda_1 W_0) < (1 - \lambda_1 \alpha)$, and this will be allocated to different tests over time. The sequence $\{\lambda_j\}_{j=1}^{\infty}$ is defined by g_t and λ_j serves as a weak estimation of α_j. g_t can be any coordinate wise non-decreasing function (line 8 in Alg. 5). $R_j := I(p_j < \alpha_j)$ is the indicator for rejection, while $C_j := I(p_j < \lambda_j)$ is the indicator for candidacy. τ_j is the j^{th} rejection time. For each p_t, if $p_t < \lambda_t$, SAFFRON adds it to the candidate set C_t and sets the candidates after the j^{th} rejection (lines 9-10 in Alg. 5). Further, the α_t is updated by several parameters like current wealth, current total rejection numbers, the current size of the candidate set, and so on (lines 11-14 in Alg. 5). Then, the decision R_t is made according to the updated α_t (line 15 in Alg. 5).

The hyper-parameters we use for the SAFFRON procedure in online false discovery rate control of Sec. 4 are aligned with the setting in Ramdas et al. (2018). In particular, the target FDR level is $\alpha = 0.05$, the initial wealth is $W_0 = 0.0125$, and γ_j is calculated in the following way: $\gamma_j = \frac{1/(j+1)^{1.6}}{\sum_{j=0}^{10000} 1/(j+1)^{1.6}}$.