COMPSCI 660 - Final Project Report - Federated Learning for Online Learning to Rank

Dzung Pham
University of Massachusetts Amherst
Amherst, Massachusetts, USA
dzungpham@umass.cs.edu

Abstract

This report details my reproduction of the latest application of Federated Learning (FL) to the problem of Online Learning to Rank (OLTR) using the Pairwise Differentiable Gradient Descent algorithm (PDGD) [13], as well as my own experiment on assessing the privacy property of the proposed algorithm. I find that applying FL as it is to OLTR without taking into consideration the skewed nature of data ownership in information retrieval will not be able to fully protect user data, as demonstrated by a simple label guessing attack using ordinary least squares regression (OLSR). The code is available at https://github.com/dzungvpham/foltr-pdgd.

CCS Concepts: • **Security and privacy** → **Privacy- preserving protocols**; • **Information systems** → **Learning to rank**.

Keywords: federated learning, learning to rank, privacy

ACM Reference Format:
Dzung Pham. 2022. COMPSCI 660 - Final Project Report - Federated Learning for Online Learning to Rank. In *Proceedings of COMPSCI 660.* ACM, New York, NY, USA, 7 pages.

1 Introduction

As privacy becomes more recognized and emphasized by Internet users and governments alike, in recent years, some attention has been focused on designing better privacy preserving ranking algorithms for information retrieval (IR) systems. Of particular interest is the Online Learning to Rank (OLTR) paradigm, which uses Machine Learning (ML) to learn a ranker from users' queries and interactions (e.g. clicks) with a search engine and its results. To protect information about users' search activities, recent research [6, 13] has applied the Federated Learning (FL) method to OLTR as a way to allow users to get better ranking results without the need to share their data.

This report presents my reproduction results of the latest state-of-the-art attempt at devising a OLTR algorithm in the FL setting (FOLTR for short). Specifically, I reproduce the Federated Pairwise Differentiable Gradient Descent (FPDGD) algorithm [13], which learns from (simulated) user clicks using the Microsoft Learning to Rank Datasets [11]. The algorithm While my evaluation results are mostly consistent with the trends observed in the original paper, there are still some numeric differences as I wrote the code without much reference to the paper's code.

In addition to the reproduction, I also test whether this training scheme can truly protect users' data by devising methods for the IR system provider (let's call it 'server' for short) to guess the users' clicks. Under the settings adopted by the original paper, I find that the server can determine with high precision and recall which documents were clicked on by applying ordinary least squares regression (OLSR) using the documents as features and the aggregated gradients as labels. This is made possible because the server knows many importance pieces of data including the users' queries, the documents served to the users and the feature values of the documents, which allow the server to reliably infer users' clicks from the new gradients.

2 Background Knowledge

2.1 Federated Learning

FL was first introduced by researchers at Google in 2016 [7] as a method to train a shared model while leaving users' data on their devices. Typically, a central coordinator (or server for short) is in charge of sending the clients the shared model and aggregating the locally learned weights/gradients from the clients' devices. Each client will perform training on their own data locally using the model received from the server. While many variations of this paradigm have been explored, the reproduction part of this report focuses specifically on the single server and many clients setting with the Federated Averaging (FedAvg) algorithm [7]. An outline of the pseudocode for FedAvg is presented in Algorithm 1 (adapted from [7]).

Algorithm 1 FederatedAveraging [7]. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

```
Server executes:
initialize $w_0$
for each round $t = 1, 2, \ldots$ do
    $S_t \leftarrow$ (random set of $m$ clients)
    for each client $k \in S_t$ in parallel do
        $w_{t+1}^k \leftarrow$ ClientUpdate$(k, w_t)$
    $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$

ClientUpdate$(k, w)$:                    ▷ Run on client $k$
$\mathcal{B} \leftarrow$ (Split data into batches of size $B$)
for local epoch $i$ from 1 to $E$ do
    for each batch $b \in \mathcal{B}$ do
        $w \leftarrow w - \eta \nabla f(w, b)$
return $w$ to server
```

2.2 Differential Privacy

FL by itself does not completely guarantee that users' data cannot be leaked, as inferences about the data can still be made by analyzing the gradients/weights received from the users. A powerful technique called differential privacy (DP) [2, 3] is often used in FL to safeguard against such inferences with formal privacy guarantee. A definition of DP is given below:

Definition 1. Differential Privacy: A randomized mechanism $M : D \rightarrow R$ with a domain D (e.g., possible training datasets) and range R (e.g., all possible trained models) satisfies ϵ-differential privacy if for any two adjacent datasets $d, d' \in D$ and for any subset of outputs $S \subseteq D$ it holds that $Pr(M(d) \in S) \leq \epsilon Pr(M(d') \in S)$.

ϵ is the privacy budget – the lower it is, the higher the privacy. One way to achieve ϵ-DP for FedAvg is to clip the model's parameters to an appropriate L_2-norm bound and then add random noise to the parameters [8]. This approach is adopted in FPDGD, specifically with the noise drawn from an approximation to the Laplacian distribution via two Gamma distributions [9].

2.3 Online Learning to Rank with Pairwise Differentiable Gradient Descent

Learning to rank (LTR) is a machine learning task where the goal is to learn a ranking model for IR systems. Traditional LTR relies on relevance/importance labels created by human raters/annotators, whereas OLTR learns from users' interactions with the query results such as clicks. Consequently, OLTR offers considerable advantages over traditional LTR since human-created labels are not only costly to obtain but also not completely reflective of users' changing preferences. Furthermore, traditional OLTR requires users' search data to be collected in order to be labelled by humans, which increases the risk of privacy breach.

One recent OLTR algorithm called Pairwise Differentiable Gradient Descent (PDGD) [10] allows non-linear differentiable rankers to be applied to OLTR while offering improved efficiency and performance. The algorithm can learn unbiased pairwise document preferences purely from user interactions. As this report reimplements PDGD, a brief description of algorithm is given below:

Consider a ranker $f_\theta(\mathbf{d})$ that sorts documents by output score in descending order, where $\mathbf{d}$ represents the features of a query-document pair. PDGD applies a Plackett-Luce to the ranker, which defines the probability of a document given a document set D as:

$$Pr(d \mid D) = \frac{e^{f_\theta}(d)}{\sum_{d' \in D} e^{f_\theta}(d')} \tag{1}$$

Using Equation 1, a ranking of length k is generated by sampling from D k times without replacement. The probability of a ranking R given D is defined as:

$$Pr(R \mid D) = \prod_{i=1}^{k} Pr(d_i \mid D \setminus \{d_1, \ldots, d_{i-1}\}) \tag{2}$$

where d_i is the document at the i-th position in ranking R.

Given a ranking, a user may click on some document d_k while ignoring some other document d_l. Denote this click event as $d_k >_c d_l$. The gradient of f is estimated as:

$$\nabla f_\theta \approx \sum_{d_k >_c d_l} \rho(d_k, d_l, R, D) \nabla Pr(d_k >_c d_l) \tag{3}$$

where ρ is the position reweighing function to counteract the position bias effect. It's defined as:

$$\rho(d_k, d_l, R, D) = \frac{Pr(R^*(d_k, d_l, R) \mid D)}{Pr(R \mid D) + Pr(R^*(d_k, d_l, R) \mid D)} \tag{4}$$

where $R^*(d_k, d_l, R)$ is the same ranking as R except the positions of d_k and d_l are swapped.

We define the probability that d_k is clicked before d_l as:

$$Pr(d_k >_c d_l) = \frac{Pr(d_k \mid D)}{Pr(d_k \mid D) + Pr(d_l \mid D)} = \frac{e^{f(d_k)}}{e^{f(d_k)} + e^{f(d_l)}} \tag{5}$$

Hence, its gradient is:

$$\nabla Pr(d_k >_c d_l) = \frac{e^{f_\theta(d_k)} e^{f_\theta(d_l)}}{(e^{f_\theta(d_k)} + e^{f_\theta(d_l)})^2} (f'_{\theta_t}(d_k) - f'_{\theta_t}(d_l)) \tag{6}$$

Combining all these pieces together, we have Algorithm 2.

Algorithm 2 Pairwise Differentiable Gradient Descent [10].

```
**Input:** initial weights: $\theta_1$; scoring function: $f$; learning rate $\eta$.
for $t \leftarrow 1 \ldots \infty$ do
    $q_t \leftarrow receive\_query(t)$
    $D_t \leftarrow preselect\_documents(q_t)$
    $\mathbf{R}_t \leftarrow sample\_list(f_{\theta_t}, D_t)$                      ▷ Eq. 1, 2
    $\mathbf{c}_t \leftarrow receive\_clicks(\mathbf{R}_t)$
    $\nabla f_{\theta_t} \leftarrow 0$
    for $d_k >_c d_l \in c_t$ do
        $w \leftarrow \rho(d_k, d_l, \mathbf{R}_t, D)$                            ▷ Eq. 4
        $w \leftarrow w \frac{e^{f_{\theta_t}(d_k)} e^{f_{\theta_t}(d_l)}}{(e^{f_{\theta_t}(d_k)} + e^{f_{\theta_t}(d_l)})^2}$          ▷ Eq. 6
        $\nabla f_{\theta_t} \leftarrow \nabla \theta_t + w(f'_{\theta_t}(d_k) - f'_{\theta_t}(d_l))$     ▷ Eq. 6
    $\theta_{t+1} \leftarrow \theta_t + \eta \nabla f_{\theta_t}$
```

2.4 Federated Averaging Pairwise Differentiable Gradient Descent with Differential Privacy

Putting all the pieces discussed thus far together, we get the FPDGD algorithm: The server uses FedAvg to aggregate the model weights, while the clients train their local models using PDGD and apply DP before returning the weights to the server. The pseudocode is outlined in Algorithm 3.

Algorithm 3 FedAvg PDGD with Differential Privacy [13]. C: set of clients, indexed by c; B: the local interaction set; n_c: the number of local interactions.

```
**Server executes:**
initialize $\theta_0$; scoring function $f$; learning rate $\eta$
for each round $t = 1, 2, \ldots$ do
    for each client $c \in C$ in parallel do
        $\theta^k_{t+1}, n_c \leftarrow$ ClientUpdate($c, \theta_t$)
    $\theta_{t+1} \leftarrow \sum_{c=1}^{|C|} \frac{n_c}{n} \theta^c_{t+1}$

**ClientUpdate**($c, \theta$):                              ▷ Run on client $c$
for each local update $i = 1, 2, \ldots, B$ do
    $\theta \leftarrow \theta + \eta \nabla f_\theta$                       ▷ PDGD in Alg. 2
$\theta \leftarrow \theta \cdot \min(1, \frac{\Delta}{\|\theta\|})$                    ▷ Clip weights
return $(\theta + \gamma - \gamma', n_c)$ to server          ▷ $\gamma, \gamma'$ gamma noise
```

3 Reproduction

This section presents the details of my reproduction of the FPDGD algorithm. The code is publicly available at: https://github.com/dzungvpham/foltr-pdgd.

3.1 Dataset

I use the MSLR-WEB10k dataset and the MQ2007 dataset from Microsoft Research [11] like in the original paper. Both datasets are divided into 5 partitions by the authors, which we use for cross-validation (3 partitions for training, 1 for validation and 1 for testing). The features are for each query-document pair. Table 1 summarizes some basic characteristics of the two datasets. Normalization was performed on both datasets to scale their feature values to mean 0 and standard deviation 1, which helps prevent numerical issues during training.

Table 1. Summary of MQ2007 and MSLR-WEB10K datasets

	MQ2007	MSLR-WEB10K
# of queries	1,700	10,000
# of documents/query	≈ 40	≈ 125
# of features	46	136
# of relevance levels	3	5

Table 2. Reproduction parameters

	MQ2007	MSLR-WEB10K
# of clients	1,000	1,000
# of rounds	200	100
# of queries/client/round	4	2
# of documents shown	10	10
# learning rate	0.1	0.1
# (DP) sensitivity	3	3
# (DP) ϵ	1.2	1.2

3.2 Setup

I write Python code from scratch to simulate the FL setting and the OLTR environment. Numpy [5] and Scipy [12] are extensively used for all computations to avoid numerical issues (e.g. overflowing). Table 2 summarizes the main parameter values of the reproduction. Due to resource constraints, cross-validation was performed only once using the optimal hyperparameters chosen the the original paper. The ranker is a linear model with no bias.

3.3 Click Models

I simulate users' click behavior with the Cascade Click Model (CCM) [4], where a user is assumed to examine a query result page sequentially from top to bottom, with a clicking probability $Pr(click|r)$ for a document with relevance r and a stopping probability $Pr(stop|click, r)$ after a click on a document with relevance r. There are three common CCM models: a *perfect* user who looks for as many highly relevant documents as possible, a *navigational* user who looks for a few decently relevant documents, and an *informational* user who accepts even completely irrelevant documents. Consequently, the informational model results in the most number of click-not clicked pairs while the navigational models results in the lowest number of pairs. Table 3 shows the values of $Pr(click|relevance)$ and $Pr(stop|click, r)$ that are used.

Table 3. CCM settings for MSLR-WEB10K (MQ2007) dataset

	perfect	navigational	informational
$Pr(click\|r)$			
$r = 0$	0.0 (0.0)	0.05 (0.05)	0.4 (0.4)
$r = 1$	0.2 (0.5)	0.3 (0.5)	0.6 (0.7)
$r = 2$	0.4 (1.0)	0.5 (0.95)	0.7 (0.9)
$r = 3$	0.8 (−)	0.7 (−)	0.8 (−)
$r = 4$	1.0 (−)	0.95 (−)	0.9 (−)
$Pr(stop\|c, r)$			
$r = 0$	0.0 (0.0)	0.2 (0.2)	0.1 (0.1)
$r = 1$	0.0 (0.0)	0.3 (0.5)	0.2 (0.3)
$r = 2$	0.0 (0.0)	0.5 (0.9)	0.3 (0.5)
$r = 3$	0.0 (−)	0.7 (−)	0.4 (−)
$r = 4$	0.0 (−)	0.9 (−)	0.5 (−)

3.4 Evaluation Method

I run the algorithm on all 5 folds of each dataset, using the test partition of each fold for offline performance measurement. Online performance is measured with the ranking results served to the users with a multiplicative discount factor of 0.9995 for each succeeding query. Note that in a typical FL settings, the users selected for training are random, so unlike the original paper, I don't measure online performance for more rounds than the number of queries per round.

I use the Normalized Discounted Cumulative Gain (nDCG) to the measure the effectiveness of the ranker for both offline and online performance. Specifically, I use the following formula:

$$nDCG@10 = \frac{DCG@10}{\text{Ideal } DCG@10}$$

where DCG is defined as:

$$DCG@10 = \sum_{i=1}^{p} \frac{2^{r_i} - 1}{\log_2(i+1)}$$

r_i represents the relevant of the document at position i in the ranking results, and ideal DCG is the DCG of the best ranking possible.

3.5 Results

Figure 1 and 2 show the average cross-validation offline and online performance of the linear ranker on the MSLR-WEB10K and MQ2007 datasets with different click models. The following observations can be made:

- **Offline performance:** With MSLR-WEB10K, test nDCG @10 reaches ≈ 0.375 for all three click models after 100 rounds of training. With MQ2007, test nDCG@10 reaches ≈ 0.44 for all three click models as well. MSLR-WEB10K's performance trajectory exhibits more variability the noisier the click model becomes, whereas MQ2007 doesn't show as much variability.
- **Online performance:** For both datasets, online nDCG @10 differs significantly between the click models, with perfect having the highest performance, navigational the second highest and informational the lowest. With MSLR-WEB10K, online performance are all worse than offline performance, but with MQ2007, only the informational model has a lower online performance than offline. It should be noted that online performance is influenced by the local training done by each client, since after each query, the local model is updated and hence the ranking behavior is different across the clients.
- **Effect of click models:** The noisier the click model is, the worse the online performance becomes. The gap in online performance between perfect and navigational is smaller than that between navigational and informational. However, the offline performance stays consistent across the click models. Offline performance trajectory shows greater variability the noisier the click model is for MSLR-WEB10K but not for MQ2007.
- **Comparison with original paper's results:** While the exact values of the measurement differs, the trends are consistent between my reproduction and the original paper. Specifically, both shows a reduction in online performance when the click model becomes noisier as well as a consistent offline performance regardless of the click models. My reproduction has a higher offline performance for MSLR-WEB10K but lower for MQ2007 than in the original paper.

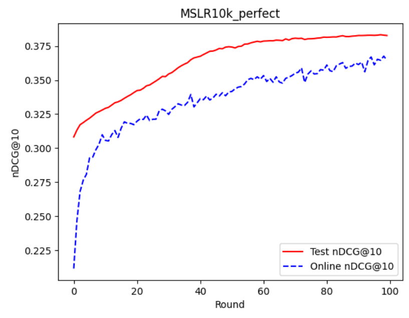
The difference in FPDGD's performance behavior between MSLR-WEB10K and MQ2007 is likely attributable to the different characteristics of the two datasets (see Table 1), especially in terms of number of queries, number of documents per query, and number of features. Since MQ2007 is relatively much smaller than MSLR-WEB10K, each query-document pair has a higher chance of being used for training and hence the likelihood of overfitting is higher. This might explain why the perfect and navigational model has higher online performance than offline for MQ2007.

4 Privacy Experiment

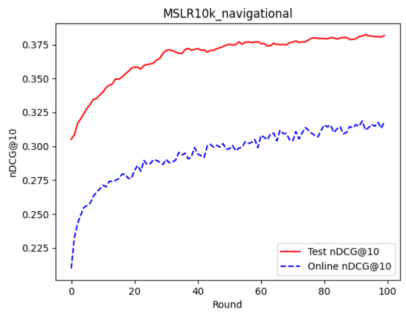
4.1 Methodology

In addition to the reproduction, I also experiment with guessing what a user has clicked on based on the model weights received from the user. The goal is to test if FPDGD can truly protect users' data as it claims. Let f' denote the client's model, f denote the global model, R denote the ranking results, and d denote any query-document pair. I come up with the following heuristic attacks:

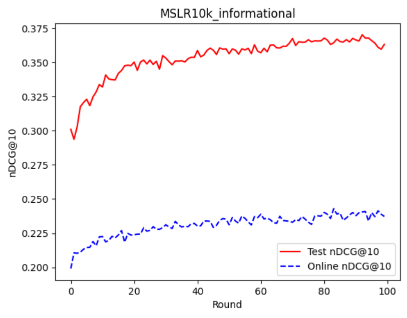
- **Higher score:** If $f'(d) > f(d)$, then guess that d is clicked
- **Higher probability:** If $\frac{e^{f'(d)}}{\sum_{r \in R} e^{f'(r)}} > \frac{e^{f(d)}}{\sum_{r \in R} e^{f(r)}}$, then guess that d is clicked
- **Higher distance count:** If $[\sum_{d' \in R, d' \neq d} \mathbb{1}(f'(d) - f'(d') > f(d) - f(d'))] > 5$, then guess that d is clicked



(a) Perfect

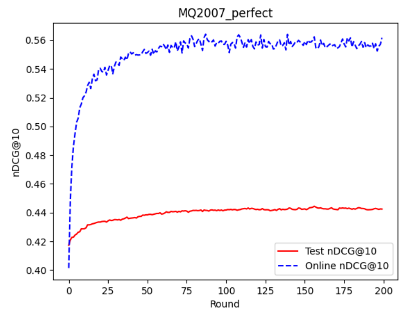


(b) Navigational

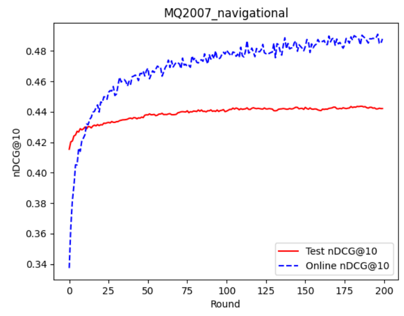


(c) Informational

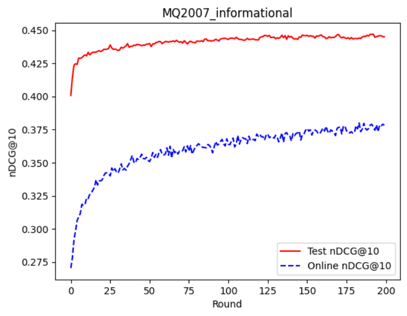
Figure 1. Offline and online performance on the MSLR-WEB10K dataset



(a) Perfect



(b) Navigational



(c) Informational

Figure 2. Offline and online performance on the MQ2007 dataset

- **Linear Regression:** I treat the received weights as the labels and the documents in the ranking shown to the users as features (i.e. feature k of the attack model is all the features in document k), then run OLSR without any bias term and guess the documents with positive OLSR weights. This is because for a linear ranker, the FPDGD algorithm (see Algorithm 2) is simply performing a weighted sum of the documents' features, where clicked documents have a positive weight and not clicked documents have a negative weight, then adding DP noise to the results. In other words, the client's weights are a linear combination of the documents' features combined with some random noise, which makes this a suitable problem for OLSR.

I run the attacks with the models trained in the perfect click model scenario. The user will be given a deterministic ranking based purely on the model scores instead of a sampled ranking, and will follow all three click models. Unlike in the training stage, the model weights are returned to the server after one query instead of multiple queries. For each data fold, I use the validation partition that is not used in the reproduction step, thus ensuring that the data has not been seen before. The performance metrics are averaged across all queries in the validation set and across all 5 folds.

4.2 Results

Table 4 showcases the average accuracy, precision and recall of the different attacking methods on MSLR-WEB10K and MQ2007. The performance of randomly guessing 3 or 4 items (which is around the same number of predictions of the attack methods) is also included for reference. As can be seen, OLSR outperforms all other attacks by a significant margin in all metrics and all click models. While its recall and accuracy is fairly consistent across the different click models, its precision is noticeably lower for the navigational model.

4.3 Discussion

The OLSR attack shows that FPDGD can leak information about users' activities even with DP applied. This loophole is possible due to the following characteristics of IR systems in this FL setting:

- The users only own the clicks.
- The server owns the documents, the queries and the features used for training.
- The server also knows which documents are served to the users and potentially also the ordering.

Table 4. Average performance of label guessing attacks on MSLR-WEB10K (MQ2007)

Click model	Accuracy	Precision	Recall
Random 3			
Perfect	0.59 (0.57)	0.26 (0.33)	0.30 (0.30)
Navigational	0.64 (0.64)	0.16 (0.14)	0.30 (0.30)
Informational	0.58 (0.58)	0.32 (0.29)	0.30 (0.29)
Random 4			
Perfect	0.55 (0.53)	0.26 (0.33)	0.40 (0.40)
Navigational	0.57 (0.57)	0.16 (0.14)	0.39 (0.39)
Informational	0.54 (0.55)	0.32 (0.31)	0.40 (0.41)
Higher score			
Perfect	0.53 (0.56)	0.30 (0.40)	0.52 (0.58)
Navigational	0.55 (0.56)	0.25 (0.24)	0.65 (0.73)
Informational	0.57 (0.57)	0.43 (0.41)	0.67 (0.68)
Higher prob.			
Perfect	0.63 (0.60)	0.35 (0.44)	0.33 (0.48)
Navigational	0.71 (0.69)	0.35 (0.35)	0.50 (0.66)
Informational	0.67 (0.65)	0.54 (0.51)	0.48 (0.58)
Higher dist.			
Perfect	0.53 (0.56)	0.30 (0.39)	0.57 (0.61)
Navigational	0.56 (0.56)	0.22 (0.20)	0.72 (0.75)
Informational	0.60 (0.59)	0.41 (0.39)	0.73 (0.71)
OLSR			
Perfect	**0.88 (0.88)**	**0.75 (0.80)**	**0.94 (0.93)**
Navigational	**0.81 (0.84)**	**0.57 (0.63)**	**0.93 (0.94)**
Informational	**0.85 (0.88)**	**0.72 (0.77)**	**0.92 (0.94)**

In a typical FL setting, the users are the sole owner of everything related to their data, but with IR, the server also owns a significant part of the data. This makes it feasible for the server to infer the document clicks even with strong DP guarantee. The skewed nature of data ownership in IR highlights the potential risk of applying FL without fully investigating all potential leakage channels.

To defend against this kind of privacy attack for FOLTR, one method is to disassociate the gradients/weights from individual users, which can be achieved via a secure weight aggregation mechanism for FL [1]. It might still be possible for the server to determine which documents are clicked on, but it will be more difficult for the server to associate the clicks with any particular user.

5 Conclusion

In summary, this report demonstrates the reproducibility of the FPDGD algorithm as well as highlights the risk of privacy leakage when applying FL to OLTR due to the skewed data ownership. FOLTR is a relatively new research direction, so it's important that all major security/privacy risks are identified and understood. Following this report, there are several next steps that can be taken:

- Investigate secure aggregation for FOLTR and/or develop a different privacy protection mechanism that doesn't rely on secure aggregation.
- Investigate poisoning attack (e.g. for document promotion).

Acknowledgments

I would like to thank Professor Amir Houmansadr for suggesting the idea for this project.

References

[1] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1175–1191. https://doi.org/10.1145/3133956.3133982

[2] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Third Conference on Theory of Cryptography* (New York, NY) *(TCC'06)*. Springer-Verlag, Berlin, Heidelberg, 265–284. https://doi.org/10.1007/11681878_14

[3] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3–4 (aug 2014), 211–407. https://doi.org/10.1561/0400000042

[4] Fan Guo, Chao Liu, and Yi Min Wang. 2009. Efficient Multiple-Click Models in Web Search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining* (Barcelona, Spain) *(WSDM '09)*. Association for Computing Machinery, New York, NY, USA, 124–131. https://doi.org/10.1145/1498759.1498818

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[6] Eugene Kharitonov. 2019. Federated Online Learning to Rank with Evolution Strategies. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (Melbourne VIC, Australia) *(WSDM '19)*. Association for Computing Machinery, New York, NY, USA, 249–257. https://doi.org/10.1145/3289600.3290968

[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Jerry Zhu (Eds.). PMLR, 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html

[8] Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/pdf?id=BJ0hF1Z0b

[9] Vaikkunth Mugunthan, Anton Peraire-Bueno, and Lalana Kagal. 2020. PrivacyFL: A Simulator for Privacy-Preserving and Secure Federated Learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 3085–3092. https://doi.org/10.1145/3340531.3412771

[10] Harrie Oosterhuis and Maarten de Rijke. 2018. Differentiable Unbiased Online Learning to Rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Torino, Italy) *(CIKM '18)*. Association for Computing Machinery, New York, NY, USA, 1293–1302. https://doi.org/10.1145/3269206.3271686

[11] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013). http://arxiv.org/abs/1306.2597

[12] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[13] Shuyi Wang, Bing Liu, Shengyao Zhuang, and Guido Zuccon. 2021. Effective and Privacy-Preserving Federated Online Learning to Rank. In *Proceedings of the 2021 ACM SIGIR International Conference on Theory of Information Retrieval* (Virtual Event, Canada) *(ICTIR '21)*. Association for Computing Machinery, New York, NY, USA, 3–12. https://doi.org/10.1145/3471158.3472236