Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Privacy-preserving Federated Learning and its application to natural language processing

Balázs Nagy [a,*], István Hegedűs [a], Noémi Sándor [a], Balázs Egedi [a], Haaris Mehmood [b], Karthikeyan Saravanan [b], Gábor Lóki [a], Ákos Kiss [a]

[a] Department of Software Engineering, University of Szeged, Hungary
[b] Samsung Research, London, United Kingdom

ARTICLE INFO

Article history:
Received 8 July 2022
Received in revised form 9 March 2023
Accepted 10 March 2023
Available online 14 March 2023

Keywords:
Federated Learning
Local differential privacy
Randomized response
NLP
String hashing

ABSTRACT

State-of-the-art edge devices are capable of not only inferring machine learning (ML) models but also training them on the device with local data. When this local data is sensitive, privacy becomes a crucial property that must be addressed. This implies that sharing data with a server for training a model is undesirable and should be avoided. The Federated Learning (FL) approach can help in these situations, however, FL alone is still not the ultimate tool to solve all challenges, especially when privacy is a major concern. We propose a privacy-preserving FL framework, which leverages the concepts of bitwise quantization, local differential privacy (LDP), and feature hashing for input representation in the collaborative training of ML models. In our approach, the local model updates are first quantized, then a randomized-response technique is applied on the resulting update vector.

Although our proposed framework functions with arbitrary types of input features, we emphasize its usability with natural language data. The text input on the client-side is encoded using a rolling-hash-based representation, which provides a combined solution for the high resource demands of embedding algorithms and the privacy concerns of sharing sensitive data. We evaluate our method in a sentiment analysis task using the IMDB Movie Reviews dataset as well as a rating prediction task with the MovieLens dataset augmented with additional movie keywords. We demonstrate that our approach is a feasible solution for private language processing tasks on edge devices without the use of resource-hungry language models or privacy-violating collection of client data.

1. Introduction

Machine learning (ML) applications have become exceedingly widespread in the past few years, and the technology nowadays is an essential part of our everyday life. There is a wide spectrum of appealing applications in many fields, including but not limited to healthcare diagnostics, chatbots, and other artificial intelligence-based (AI) agents, e.g., in public administration, or the various tools for image processing on mobile devices, just to name a few.

In recent years, privacy has evolved into an invaluable property, which has also been strictly regulated by law, such as the GDPR[1] in the European Union. However, there are many ML-based applications that rely on protected, classified, or sensitive data that cannot be shared with any data curators, trusted or not. At the same time, this data is an essential part of the personalized ML-based applications and needed for the training procedure to learn proper and accurate knowledge from user data. One possible solution to this antagonism is to keep the data at the user's end and train models locally at the edges of the greater network. This is the basic concept of the Federated Learning (FL) approach [1,2], where only updates of the locally trained models are communicated with a server, but the raw training data itself is never seen by it. With the rapid spread of edge devices, e.g., Internet of Things (IoT) technology, smart home gadgets, or mobile devices, there is an ongoing demand for decentralized and distributed training approaches. A frequently occurring example for a typical FL application is the distributed training of a common global model used by several institutions or companies who do not want to, or are not allowed to share their local data with each other [3]. More specifically, the global model can be imagined as a healthcare application that is trained by the participating hospitals using their locally stored patient data [4].

Despite the efforts of keeping the data safe locally at the edge of the network, leakage can still occur [5,6]: the training data can be guessed via reverse engineering from the local model weights communicated with the server. Local differential privacy (LDP) [7] is one possible way to overcome this issue, and it indeed

* Corresponding author.
E-mail address: bnagy@inf.u-szeged.hu (B. Nagy).

1 https://gdpr.eu

works very well in many practical applications [8] even in an FL setting [9].

Training most ML applications requires numerical data, which means that a picture, a piece of text, or any kind of unstructured input cannot be fed directly into these models. Instead, a feature extraction step has to precede the actual training process. In those cases where the input features are encoded in the input itself, such as the pixel intensities of images, no extra feature extraction step is needed. For text, the usual pre-processing step is the embedding procedure: a word, a sentence, or even a bigger chunk of the textual input, such as a full document, is converted into a numerical feature vector. There are many such embedding techniques available, each having its own advantages and disadvantages. The current state-of-the-art approaches in this field are huge language models with attention-based transformers, such as BERT [10] or GPT-3 [11] capable of solving challenging tasks in Natural Language Processing (NLP) with unprecedented accuracy.

However, achieving a reasonable level of privacy in text representation techniques is not at all straightforward. This is particularly true for applications involving models trained on a server using data collected directly from the clients. For new word predictions and simple term frequency estimations, where data can be represented with a fixed-size vocabulary, one can easily apply any randomized-response-based algorithm, such as RAPPOR from Google [12] or the count-mean sketch algorithm by Apple [13]. On the other hand, the common practice of adding random noise to either bag-of-words-like one-hot representations or to the dense, real-valued word embedding vectors in a more general NLP task involving arbitrary free text is not naturally applicable in text-based model training scenarios. As a result of the perturbation, the meaning of the words and, consequently, the information encoded in the input text would most certainly be lost [14].

It is possible to introduce some level of privacy to the dense embeddings by perturbing the input text directly while keeping the original semantics. The metric differential privacy method [15] leverages the hierarchy in the semantics of words and perturbs the vector representation to obtain a new word with a more general semantic meaning of the original input. That is, the method changes the syntactics of the input by finding a new word in the semantic space that is in the vicinity of the original word.

Privacy can be further improved locally by applying hashing techniques in the input text representation step. The hashing algorithms are often used for feature vectorization and dimensionality reduction of large feature spaces [16] including natural language text data. An appropriate hash function provides a one-way only transformation of its input, which means that from a given hash value it is very difficult to get back the original input. In the case of string or text hashing, a brute force search, i.e., hashing the complete vocabulary for the target language and finding the corresponding input for a given hash value, would be an obvious choice to violate the privacy of the method. This could work in many cases, although, on the one hand, the complete vocabulary for free text is not known for absolute certainty, and, on the other hand, due to the limited size of hash values, which is practically smaller than the size of the actual vocabulary in most applications, hash collision can further obscure the results. The above two aspects ensure that the original text input is highly unlikely to be found without doubt. Hence, encoding a textual input via a string hashing algorithm ensures a straightforward way of maintaining privacy to a reasonable extent. Nevertheless, it is important to note here that the privacy introduced by the hashing is only an extra benefit and does not contribute to the privacy budget of our LDP approach. Neither the raw text input nor its hashed output is part of the update that is sent to the server in the communication rounds. As shown in our present study, the hashed text features can be utilized in training ML models both centrally and distributively, where the models partially or fully rely on textual input without a significant loss of accuracy. Being a simple function call, the method is low-cost, very fast, and therefore extremely beneficial to use on limited-resource edge devices.

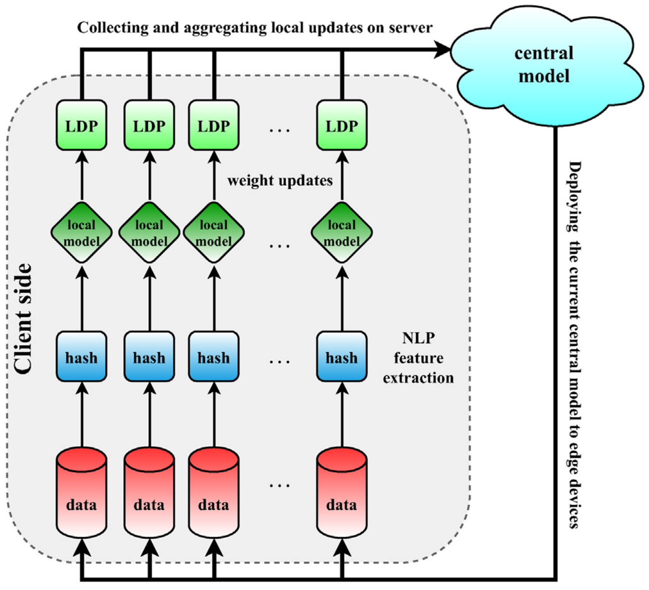

Collecting and aggregating local updates on server
central model
LDP
weight updates
local model
Client side
hash
NLP feature extraction
data
Deploying the current central model to edge devices

Fig. 1. High-level overview of our proposed framework.

In this study, we present a privacy-preserving FL framework to train ML models using local data including natural language text at the edge of the network. A high-level overview of the proposed system is shown in Fig. 1. In our setting, the weight differences between the local and server models are used for the update. In order to ensure privacy, randomized-response-based local differential privacy is applied, where noise is added to the quantized weight differences prior to communicating them with the server. The local updates are then corrected for the noise and averaged on the server-side to obtain a new server model. Locally at the edge, a vocabulary-less rolling string hashing technique is used to represent the textual features for training the local model. This method allows us to avoid the use of resource-heavy language models, or collecting user statistics to obtain an embedding model with a reduced and optimized user-specific vocabulary, which would clearly violate privacy. The quantization of the weight differences together with the hashed text representation further improve the privacy of our framework beyond the sole application of the LDP technique.

Although the key idea behind our framework is to work equally well for any feature types, we evaluate our proposed approach mainly in NLP-based tasks, where the client data is text. These tasks, in our view, are good illustrations of real-life applications, where the ML models are to be personalized with user data. We chose a binary sentiment analysis task of movie reviews using only textual input, and a movie rating prediction task using both categorical and NLP features. We compare the performance of our method with that of centrally trained models employing widely used embedding techniques, including the word2vec model [17] trained specifically on the utilized movie reviews corpus, and the pre-trained embedding model fastText [18].

The essential advantages of our proposed locally differentially private FL framework are:

1. Our framework, in principle, can be applied to any arbitrary input feature. We show example use cases here with

pure natural language text inputs as well as mixed feature spaces with text along with numerical and categorical features.

2. In our FL environment, LDP is ensured by a client-side randomized response technique applied on the quantized model weight updates.
3. Reduced client to server communication cost due to the bitwise quantization of weight updates.
4. Textual features can easily and privately be included with a low-cost hashing-based embedding approach.

We note here that the primary intent of this study is to provide a generally applicable framework capable of performing model training on user devices with local user data while keeping this data private. We apply several approximations to this end, and first and foremost investigate the trade-off between accuracy and privacy in example ML applications involving textual input. Thoroughly reviewing the current trends and state-of-the-art in large-scale NLP systems, sentiment analysis or text categorization fields is out of the scope of this study.

The rest of the paper is structured as follows: In the next section, we discuss the background of our framework and the related research studies. Section 3 presents the theoretical and algorithmic background of the methods used in our experiments. In Section 4, the details of the experiments are described. We discuss the results in Section 5, and finally, Section 6 concludes the paper.

2. Background and related work

2.1. Federated learning

FL is a novel technology with the purpose of training machine learning models locally on edge devices without moving the actual training data to a server. Instead of the data, the model itself or some parts of it are moved and trained collaboratively with local data stored on the client devices. Recent literature [4] includes well-detailed surveys of the FL technology and its applications. Briefly, FL is a parameter server-like approach for the distributed training of machine learning models over a set of clients [1]. Considering its architecture, our proposed FL framework can be viewed as a Horizontal FL according to the definition of Yang et al. [2], where the feature space is the same among the clients but the data varies. In this setting, the clients store the training data locally, and a central server periodically sends the current version of the machine learning model to the clients. Each client performs an update on the received model using its local subset of the training data, and sends back the updated model instance to the server. Finally, in each iteration, the server aggregates the individual client models to obtain a new server model. The advantage of this framework is the support of data privacy, since the raw local data stored by a client is never sent to any other client device or to the server. In addition, the model update is performed by the clients locally, which means that the machine learning process, i.e., the training is distributed by design.

2.2. Privacy preservation

A widely used approach to achieve data privacy in many statistical data analyses and machine learning applications is the concept of differential privacy (DP) [19], where an appropriate Gaussian or Laplacian noise is added to the input [20,21], gradient [22,23] or objective function [24,25]. In our present study, we focus on a local alternative to DP, LDP [26], in particular, the randomized response (RR) method [27]. It was developed for collecting statistical information privately from structured surveys. Intuitively, the method can be imagined as telling the truth based on the outcome of a coin flip, i.e., the individual participant in a survey tells the truth with a predefined probability or lies otherwise [28]. RR is directly applicable to binary or categorical data [29], but for floating point numbers additional changes are required. The RR technique is often used for counting occurrences of hashed or binary features, e.g., in Google's RAPPOR [12] and a similar solution from Apple [13].

The RR algorithm can easily be introduced into the FL context, which gives us the opportunity to apply local privacy preservation at the network edges. In particular, RR can be applied to the parameter updates of the local models trained with on-device local data before the actual update is sent to the aggregation server. A similar solution [9] has been presented recently for image processing applications, however, in our proposed setting, we transfer the data to the server as *bits* in order to reach better communication efficiency. Other alternatives aiming at privacy preservation in a FL setting have also been investigated. Recently, Zhang et al. [30,31] investigated a data privacy preserving FL framework for intelligent machinery fault diagnostics. For recent surveys on further approaches and use cases the reader is referred to the comprehensive works of Mothukuri et al. [32] and Truong et al. [33], and the references therein.

2.3. Quantized binary representation

The bitwise transfer of updates requires the binary quantization of the parameter vectors. A straightforward option is to transform the common floating point representation of the parameters to an integer-like representation, preferably a binary one. Therefore, a vector quantization approach [34] is applied on the client-side that transforms the real-valued model weight updates into a binary vector. The quantization step has a double benefit: the reduced cost in the client to server communication due to the compressed update size, as well as the straightforward application of the RR method on the binary vectors to preserve privacy. Virtually, the perturbed bits in the quantized representation of the weight differences are used to update the server model in our federated learning framework. Based on the main concept of the RR method, the original values can be recovered approximately by applying the appropriate correction on the server-side. Although this is clearly an approximation of the original weights, we show that it gives satisfactory results even for a limited number of clients.

2.4. Differential privacy

Various anonymization techniques can be applied to data sets in order to get a publicly releasable version. Nevertheless, not all of them can guarantee that the data no longer contains sensitive personal information or that it cannot be de-anonymized, or that users, i.e., information sources, cannot be identified unambiguously by combining the data with other sources [35].

To tackle this problem, differential privacy has been introduced [28]. DP is rather a definition of the concept and not a conventional algorithm, and it provides a mathematical guarantee for the amount of information leakage, controlled by a parameter ϵ. A more precise description of DP is:

Differential privacy. A potentially randomized query Q on a domain $\mathbf{D}$, $Q : \mathbf{D} \to \mathbf{R}^d$ is ϵ-differentially private if and only if

$$\forall x \in \mathbf{R}^d : P[Q(D) = x] \leq e^{\epsilon} \cdot P[Q(D') = x], \tag{1}$$

for all neighboring databases $D, D' \in \mathbf{D}$ differing in a single entry.

DP can be guaranteed by adding properly generated Gaussian or Laplacian noise to perturb the input, gradient, or the objective function. Furthermore, the RR technique can also be applied to ensure DP [26].

A more rigorous form of DP is the local variant, LDP, where it is assumed that the data collector is not trusted at all, and users must perturb their data locally prior to the communication with the collector. More formally

Local differential privacy. A potentially randomized query Q on a domain $\mathbf{D}$, $Q : \mathbf{D} \rightarrow \mathbf{R}^d$ satisfies ϵ-local differential privacy if and only if

$$\forall x \in \mathbf{R}^d : P[Q(v) = x] \leq e^{\epsilon} \cdot P[Q(v') = x], \tag{2}$$

for any pair of input values $v, v' \in \mathbf{D}$.

2.5. Hashed representation of text

A comprehensible format of textual data for a machine learning model is a numerical vector or matrix. Thus, a number of vectorization approaches have been introduced based typically on large embedding tables where each row of the table corresponds to a unique embedding vector of the given input text. The simple bag-of-words representation is one of the most straightforward embedding techniques. First we count all unique words in the target text corpus, i.e., create a vocabulary for the given corpus. This way we can use the vocabulary index to define a d-dimensional one-hot vector for a word from the corpus, where d is the total number of unique words. When the aim is to embed a sequence of words, e.g., sentences or paragraphs of a document, the so-called term frequency matrix can be constructed by counting the individual word frequencies for each sequence. This is a sparse integer matrix of size $n \times d$, where n is the number of sequences in the document, and d is the total number of unique words for the entire document, i.e., the vocabulary. A matrix element (i, j) corresponds to the frequency of the word $j \in d$ in the sequence $i \in n$. This embedding approach does not encode any contextual information of a given word. This information can be introduced by using pre-trained embedding models. Instead of the high-dimension sparse vectors, shallow models such as the word2vec [17] model, GloVe [36], or fastText [18] can be used to obtain a fixed dimensional dense representation of words (or even larger chunks of text). These methods, however, require the storage of the embedding table together with the vectors in addition to the vocabulary. On the other hand, the most recently developed deep language models, such as BERT [10] or GPT-3 [11], apply so-called Transformers with multiple attention layers [37], and they require an additional storage of the hundreds of millions of model parameters.

The utilization of large language models or high-dimensional pre-trained embedding techniques is often impractical on edge devices due to resource limitations. There are numerous attempts to circumvent this issue by utilizing compression approaches, which can accelerate model inference in lower-resource environments and enable the economically efficient and optimized execution of AI-based applications. Compression methods such as pruning, quantization, knowledge distillation, parameter sharing, tensor decomposition, and sub-quadratic complexity transformers essentially rely on the large extent of redundancy among the weights of large neural models. Deep diving into the exact details behind these methods is outside the scope of this study. Interested readers are referred to a recent comprehensive review [38] and references therein.

Another widely used and effective approach to reduce the high-dimensional feature space is feature hashing [39]. The basic idea behind this approach is to map the original large feature space into a smaller one by a hashing function. With this hash-based dimensionality reduction, collision easily becomes an issue when the original feature space is large, e.g., for huge vocabularies in NLP applications. In these cases, a single hash function is usually not enough to achieve a reasonable accuracy. Several solutions have been proposed, where two [40] or more [41,42] hash functions are used for the embedding. These methods combine the results of multiple hash functions to obtain a better representation with a reduced collision rate. It is also possible to preserve the context information of text when hashing. The recursive min-wise hashing algorithm [43] is developed for fast and computationally effective text similarity calculation using context-preserving fingerprints. Another way of preserving the contextual information is to apply hashing to the word co-occurrence matrix [44]. The only limitation of both methods mentioned above is that the text has to be first represented with a bag-of-words-like approach or a word co-occurrence matrix, which requires the storage of a vocabulary.

Most recently, the deep hash embedding method [45] has been proposed, where the text input is first encoded to a dense representation with multiple hash functions, and then a deep neural network is applied to obtain the final embeddings.

3. Methods

In this section, we describe in detail how our proposed privacy-preserving FL system operates, and we show the algorithmic details of the client- and server-side implementations of our RR-LDP approach. We also discuss the hashing-based text representation method used on the client-side to further enhance privacy and encode the local textual input with a low-cost function execution.

3.1. Federated RR-LDP

The server- and client-side operations in our FL framework are shown in Algorithms 1 and 2, respectively.

In our federated learning implementation, the local devices receive the model parameters from the server and perform the model update with the local data as usual. As mentioned above, our framework corresponds to a Horizontal FL architecture, and accordingly, there is no data exchange among the clients at any point. After the local update, the clients send back the differences between the newly trained and the originally received parameter sets (line 7 in Algorithm 2) after applying a quantization step (line 8 in Algorithm 2) and an LDP privatization (line 9 in Algorithm 2).

Computing the differences allows us to easily control the size of the parameter values, since they are proportional to the learning rate. A probabilistic quantization is applied to the parameter differences to transform the values into the set of {0, 1} as defined in (3).

$$[v] = \begin{cases} 0 & \text{with probability } (q - v)/2q \\ 1 & \text{with probability } (v + q)/2q, \end{cases} \tag{3}$$

where $[v]$ is the quantized value of $v \in \Delta_i$, where $\Delta_i = (\Theta^i_{t+1} - \Theta_t)$, that is, the difference between the locally updated model weights at client i, Θ^i_{t+1} and the server weights, Θ_t in the communication round t. Each value v was first clipped to the range $[-q, q]$. The quantization is followed by the randomized response step applied to the binary data $[v]$:

$$[\tilde{v}] = \begin{cases} [v] & \text{with probability } \frac{e^{\epsilon}}{1+e^{\epsilon}} \\ 1 - [v] & \text{with probability } \frac{1}{1+e^{\epsilon}}, \end{cases} \tag{4}$$

where ϵ is the privacy parameter.

As an example, consider the vector $[-0.154, 0.761, \ldots, -0.433]$. Let the quantization range, q equal to 0.5, so the vector components will be clipped to the range $[-0.5, 0.5]$, which gives the clipped vector of $[-0.154, 0.500, \ldots, -0.433]$. A probabilistic binary quantization step with the probabilities shown in (3) gives a binary vector of $[1, 1, \ldots, 0]$. Finally, the bits in the binary vector are flipped according to (4) resulting in the final perturbed vector $[1, 0, \ldots, 0]$, which is then sent back to the server. A naturally arising additional advantage of this method is the low upload bandwidth, since the values of the parameter vector are still binary.

At the server-side, the received parameters are averaged to get an approximation to the actual update step. Before applying the update, the bias introduced by the randomized response should be removed and the values have to be de-quantized to their original range. The set of updates, Δ, can be restored via (5) (line 8 in Algorithm 1).

$$\Delta = \left([\widetilde{\Delta}] - \frac{1}{1+e^{\epsilon}}\right) \cdot \frac{e^{\epsilon}-1}{1+e^{\epsilon}} \cdot 2q - q \tag{5}$$

The resulting values can be applied to the server model as a weight update.

Our framework can be considered as an extension of the traditional FL method with two additional components, namely a randomized-response-based local privacy-preserving method and a hash-based textual data representation (in NLP use cases). The approach is similar to the one applied in an image classification scenario by Sun et al. [9], however, we added an extra step, namely, the binary quantization of the local updates prior to the randomized response perturbation. A further difference between our method and that of Sun et al. [9] is the definition of the client-side update: while in our study we define the updates as differences between the updated local and server weights, Sun et al. use the actual updated local weights and apply splitting and shuffling steps to them. Note that the loop at line 3 in Algorithm 1 is executed in parallel for all nodes independently of each other.

Algorithm 1 Server

```
 1: $\Theta_0, q, \epsilon \leftarrow$ initalize()
 2: loop t=0, 1, ...
 3:     for every node $i$ do
 4:         send $(\Theta_t, q, \epsilon)$ to $i$
 5:         receive $[\widetilde{\Delta}_i]$ from $i$
 6:     wait()
 7:     $[\widetilde{\Delta}] \leftarrow \sum_{i\in K} \frac{|N_i|}{|N|}[\widetilde{\Delta}_i]$
 8:     $[\Delta] \leftarrow$ correction($[\widetilde{\Delta}]$)
 9:     $\Delta \leftarrow$ dequantize($[\Delta]$)
10:     $\Theta_{t+1} \leftarrow \Theta_t + \Delta$
```

Algorithm 2 Client (at device i)

```
 1: data$_i$: local data on device $i$
 2: $x_h \leftarrow$ hash_text(data$_i$)
 3: $x_e \leftarrow$ extract_feature(data$_i$)
 4: $x_i \leftarrow [x_h, x_e]$                    ▷ $x_e$ is optional
 5: procedure onReceive($\Theta_t, q, \epsilon$)
 6:     $\Theta^i_{t+1} \leftarrow$ update($\Theta_t, x_i$)
 7:     $\Delta_i \leftarrow \Theta^i_{t+1} - \Theta_t$
 8:     $[\Delta_i] \leftarrow$ quantize($\Delta_i, q$)
 9:     $[\widetilde{\Delta}_i] \leftarrow$ randomize($[\Delta_i], \epsilon$)
10:     send($[\widetilde{\Delta}_i]$)
```

3.2. Text representation with string hashing

Representing text on edge devices with limited computational and storage resources is a challenging task. In this study, we adapted the polynomial rolling hash function[2] used successfully in tasks such as the Rabin–Karp string search algorithm [46]. This approach involves a simple function execution on the input string that uses multiplications and additions. The hash value H for a string s of length $|s|$ is computed as shown in (6).

$$H = \sum_{i=1}^{|s|} s_i \cdot P^i \mod M \tag{6}$$

The parameter M essentially defines the hash bucket size which is usually a large positive integer. The s_i is a small integer associated with the ith character of the input string s. These character values are obtained, e.g., by simply associating an integer to each character in the English alphabet, that is, ‘a’ = 1, ‘b’ = 2, ..., ‘z’ = 26. The parameter P is best to be chosen as a prime number close to the total number of different characters used in the alphabet. That is, there are 26 lowercase characters in English alphabet, so choosing $P = 31$ is reasonable.

We define word- and sentence-level text representations with the rolling hash function. The word-level representation is an M dimensional one-hot vector where the index of the single non-zero element is the hash value H for the input word. For example, with the bucket size $M = 5000$ and $P = 31$, the hash value $H = 2196$ can be obtained for the string “dog” via (6). Accordingly, the word-level embedding vector for “dog” will be $[0, 0, \ldots, 1, \ldots, 0]$, where the 1 is at position 2196 in the vector.

For a sentence, we compute the hash value H_w for each word w of the sentence and set the corresponding elements to 1 in an M dimensional binary vector. As an example, consider the text “cats and dogs”. Tokenizing on white space gives us a list of tokens, [“cats”, “and”, “dogs”]. For each token in this list, a hash value is computed via (6), which gives us [283, 4279, 3225] with the default settings. The M dimensional embedding vector $[0, \ldots, 1, \ldots, 1, \ldots 1, \ldots, 0]$ is then obtained, where the 1s are at positions corresponding to the computed hash values. Since it is fairly difficult to get back the input text from the hash value, we may consider these representations to be adequately private.

The resulting binary vectors can either be used directly as an input for a model or fed into a trainable embedding layer to obtain a dense real-valued representation.

We emphasis here again that the above hashing approach is used solely for a vocabulary-free and low-cost embedding purpose. The raw input and its hashed output are never communicated to the central server. The hashed text serves as the input for the local model update and it never leaves the user device during any information exchange between the local device and the server.

4. Experiments

In this section, we summarize the details of our experiments. The proposed RR-LDP-based federated learning approach with hashed NLP features is evaluated on two datasets: the IMDB Movie Reviews and the MovieLens 1M user ratings augmented with additional movie keywords text data. The IMDB Movie Reviews dataset was chosen because it uses natural language text features exclusively, and, therefore, is adequate to show the performance of our proposed method in an NLP-based setting. The other dataset, MovieLens 1M extended with keywords, on the other hand, was chosen to evaluate our method on a mixed feature set containing categorical as well as textual data. We first give a brief description of these datasets, which is followed by details about the text pre-processing steps and the machine learning models used.

[2] https://cp-algorithms.com/string/string-hashing.html

4.1. Datasets

IMDB movie reviews. The IMDB Movie Reviews dataset [47] consists of 50,000 user-written reviews (25k for training and 25k for testing) collected from the Internet Movie Database (IMDB). The dataset contains textual features only, and is labeled with `positive` and `negative` labels for a binary sentiment classification task. The dataset is well-balanced with exactly the same number of positive and negative examples both in the train and test sets.

MovieLens 1M with movie keywords. The MovieLens 1M dataset [48][3] contains one million ratings for approximately 4000 movies collected from 6000 users. In order to increase the performance of the rating prediction and to demonstrate the applicability of our hash-based text representation, we augmented the default feature set with additional textual data. In particular, for each movie, we downloaded a set of keywords (if available) from the corresponding IMDB movie description. The IMDbPY Python package[4] was utilized for this purpose. The machine learning task is a binary classification, where a negative label is assigned to a movie that is rated less than 3 points out of 5, and a positive label is assigned otherwise. The dataset is fairly balanced with a total of 575,281 positive and 424,928 negative examples.

4.2. Experimental setup

Federated learning settings. The FL environment was simulated by randomly distributing the training data over a set of clients, N, where we chose 10, 50, 100, and 250 for N, and in each experiment, every client has the same number of samples. A centrally trained version for each model was also made as a reference. In that case, the same model settings were used, however, the training data was not distributed, and only a single server-side model was trained for each task. Consequently, the client-side quantization and privatization steps as well as the server-side correction and aggregation steps were not included in these experiments.

For each client in the FL simulations, an initial server model is deployed first, which is updated in a few epochs with the client's local data. The randomized response algorithm is then applied on the client-side to the differences between the quantized weights of the deployed server and the updated local model. The private weights from the clients are then aggregated, corrected for the RR perturbation, and dequantized on the server, and, finally, a new server model is produced. We utilized the TensorFlow Federated (TFF) framework[5] [49] for these decentralized training simulations.

Text processing and feature hashing. Our hash-based representation considers only the 26 lowercase characters of the English alphabet. Therefore, all text data was converted to lowercase and filtered accordingly, i.e., all non-alphabetical characters including numbers and punctuations were removed. The target text was then tokenized on white space and fed to the hash function to obtain the word- or sentence-level representation. No further processing has been applied to the Movie Reviews dataset.

Since the frequency of the keywords was also considered in the MovieLens experiments, English stopwords (as defined in the NLTK library [50]) were filtered from the set of downloaded keywords. In our present experiments, we set the default number of hash buckets M to 5000 and the parameter P to 31, but the effects of increasing the bucket size and thus reducing hash collisions were also studied.

Machine learning models. Both datasets essentially involve a binary classification task. For the IMDB Movie Reviews dataset, we chose the logistic regression linear model and a feed-forward fully-connected neural network with the term-frequency as well as the word2vec embedding approaches as baselines. Different-sized word2vec embedding models were trained specifically for this dataset with the Gensim library [51]. In the first case, we used the 25k reviews from the training set to build a corpus, and all unique words were included in the vocabulary. This results in a word2vec model with 93,329 words and their vectors in the vocabulary. Adding the test dataset with an additional 25k reviews extends the vocabulary with only 65 new unique words, however, the training corpus for the word2vec model is twice as large, and, therefore, we expected the embeddings to be more accurate. We also defined a word2vec model with a limited size vocabulary, where only the 5000 most frequent words were considered. We used a word vector size of 300 for every model.

The linear model consists of a single classification layer with the sigmoid activation, and the stochastic gradient descent algorithm was used to minimize the binary cross-entropy loss function. Besides the linear model, a feed-forward neural network with three fully connected dense layers was also trained for the binary sentiment analysis task. The numbers of neurons in the layers were 100, 50, and 25, respectively. These layers use the rectified linear activation function (ReLu), while the classifier layer also uses a sigmoid activation. We chose Adam [52] as the optimizer, which provides very fast convergence, and binary cross-entropy as the loss function. In order to demonstrate the performance of our framework for more complex model structures, we also trained a neural model with a 50-unit long short-term memory (LSTM) [53] layer added as the first layer to the above neural network model. For evaluation metrics, we used the accuracy and the AUC (area under the receiver operating characteristics curve) measures in every experiment, which are sufficiently acceptable for class-balanced datasets.

The input data was fed to the classifier in two different ways. First, an input layer received the full-dimensional sparse binary hash vectors directly in the sentence level format. That is, each review was represented with a single M dimensional sparse vector. Second, a trainable embedding layer was added to the model, which produced a reduced dimension dense representation of the hashed input. In this case, the input was a fixed-length list of integer indices corresponding to the hash values of the words in a movie review. Reviews shorter than the fixed maximum length were padded with a value outside of the possible set of hash values, while long reviews were truncated to the fixed maximum length. The input dimension, therefore, was always of size $(M+1)$, and the embedding dimensions of 100, 200, and 300 were tested. The input length, i.e., the number of words per review was set to 300. In order to avoid overfitting during the training, dropout layers were also introduced in the model structure.

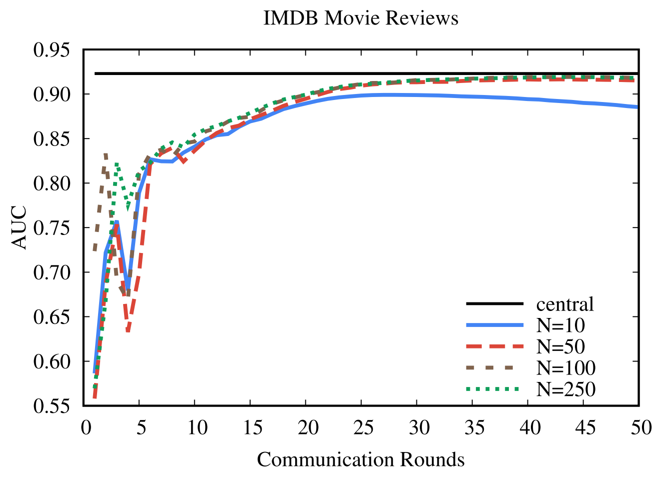
For the MovieLens dataset, we utilized the same fully connected network as above to predict the user ratings. The training data was assembled by concatenating the appropriately processed original MovieLens features with the binary hash vectors obtained for the set of keywords of a given movie. One hash vector was computed for each movie by setting the vector components to 1 according to the hash values of the keywords associated with the movie. We compare our results to a baseline where the keywords are represented with 300 dimensional pre-trained fastText [18] vectors.

5. Results and discussions

5.1. Federated RR-LDP

In our experiments, we measured the performance of the above-defined machine learning models in an FL setting, where

3 https://grouplens.org/datasets/movielens/1m/
4 https://imdbpy.github.io/
5 https://www.tensorflow.org/federated



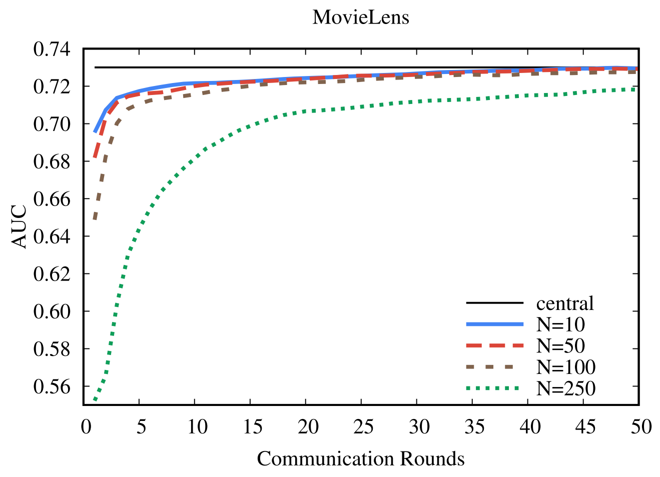


Fig. 2. Performance comparison of the central and federated RR-LDP evaluations with a varying number of clients (N) on the datasets.

Table 1
Performance of the federated RR-LDP approach with a varying number of clients (N) and communication rounds (R) on the IMDB Movie Reviews datasets. All values are AUCs.

	Number of clients			
R	$N = 10$	$N = 50$	$N = 100$	$N = 250$
1	0.586	0.558	0.724	0.570
5	0.789	0.699	0.810	0.814
10	0.841	0.836	0.847	0.854
15	0.869	0.872	0.875	0.879
20	0.889	0.895	0.899	0.899
25	0.898	0.909	0.911	0.911
30	0.899	0.913	0.916	0.915
40	0.894	0.916	0.919	0.919
50	0.885	0.915	0.918	0.918
Central	*0.923*			

Table 2
Performance of the federated RR-LDP approach with a varying number of clients (N) and communication rounds (R) on the MovieLens 1M datasets. All values are AUCs.

	Number of clients			
R	$N = 10$	$N = 50$	$N = 100$	$N = 250$
1	0.695	0.682	0.649	0.552
5	0.717	0.716	0.711	0.644
10	0.722	0.720	0.716	0.681
15	0.723	0.722	0.720	0.699
20	0.724	0.724	0.722	0.707
25	0.725	0.726	0.723	0.709
30	0.727	0.726	0.725	0.712
40	0.728	0.728	0.727	0.715
50	0.729	0.729	0.728	0.718
Central	*0.730*			

the training dataset was distributed over a set of clients. A centrally trained model, i.e., the model that was trained on the complete, undivided dataset and without applying the quantization and RR-LDP privatization steps was used as the comparison baseline. In the training procedures on the client devices, we applied the Adam optimizer with learning rates $\eta = 0.001$ for the IMDB dataset and $\eta = 0.01$ for the MovieLens dataset. At the center node, i.e., the server of the FL system, we used the stochastic gradient descent (SGD) method with a learning rate of 1, to achieve the appropriate averaging of the uploaded model parameters from the edge devices. The natural language text input in these experiments was represented with the string hashing approach using the sentence-level vector with a hash bucket size of 5000.

The performances of the FL models in terms of the AUC values can be seen in Fig. 2 and Tables 1 and 2 for the IMDB and MovieLens datasets, respectively. The results clearly show the convergence of the models with the increasing number of client devices. The performance of the models trained in the FL environment are in line with the results of the centralized baseline model, where no privacy preservation step was applied. As Table 1 and the top chart of Fig. 2 show, using 10 nodes only is not enough to reach the desired accuracy and convergence. This instability is most likely caused by the relatively high variance when averaging the parameters of too few models on the server-side. Moreover, in some cases with a larger number of edge devices ($N = 250$), slightly less accurate models are achieved for both datasets. This originates from our design choice, in particular, the same hyperparameters have been set in all FL settings. That is, we used the same learning rate as well as communication rounds and, most importantly, the local batch size has been set to get the same number of local updates. This means that the more clients there are, the smaller the local learning batches are, since the training data is distributed evenly over the clients. However, as can be seen in Fig. 2, the models still reach convergence, even in the case of the largest networks with 250 clients, where, however, a slight loss of accuracy may occur. This performance degradation implies that there is a lower-bound to the amount of training data per client, and fewer data than this yields less accurate results. Increasing the amount of training data could solve this issue.

In our privacy-preserving FL framework, we have two parameters that control the convergence and the performance of the training procedure. One of them is the quantization parameter, q, introduced in (3), which defines the scale in the transformation of the model parameters to binary values to be transferred to the server. The charts in Fig. 3 and Tables 3 and 4 show the performance of our federated RR-LDP approach with varying quantization ranges. Note the different ranges of q for the two different datasets. As can be seen, varying this value affects the performance, and the best results can be achieved when we set the quantization parameter close to the learning rate. If a high value is chosen for q, the quantization "hides" the information coded into the model parameters, and in the opposite case – choosing values that are too small – the quantization clips the information. Nevertheless, as the results also show, this information loss can be compensated by involving more edge devices in the federated training process. Since the optimal value for this parameter depends on the dataset at hand as well as the actual model architecture and learning rate, therefore, choosing the

Table 3
Effects of quantization range (q) on the performance of the federated RR-LDP for different numbers of clients (N). All values are AUCs obtained for the IMDB Movie Reviews dataset.

	Number of clients			
q	$N = 10$	$N = 50$	$N = 100$	$N = 250$
0.01	0.678	0.727	0.757	0.792
0.001	0.860	0.877	0.886	0.887
0.0001	0.887	0.903	0.900	0.891
0.00001	0.801	0.882	0.888	0.875
Central	*0.923*			

Table 4
Effects of quantization range (q) on the performance of the federated RR-LDP for different numbers of clients (N). All values are AUCs obtained for the MovieLens 1M dataset.

	Number of clients			
q	$N = 10$	$N = 50$	$N = 100$	$N = 250$
1.0	0.665	0.699	0.702	0.699
0.1	0.721	0.723	0.725	0.718
0.01	0.730	0.730	0.729	0.712
0.001	0.731	0.730	0.728	0.710
Central	*0.730*			

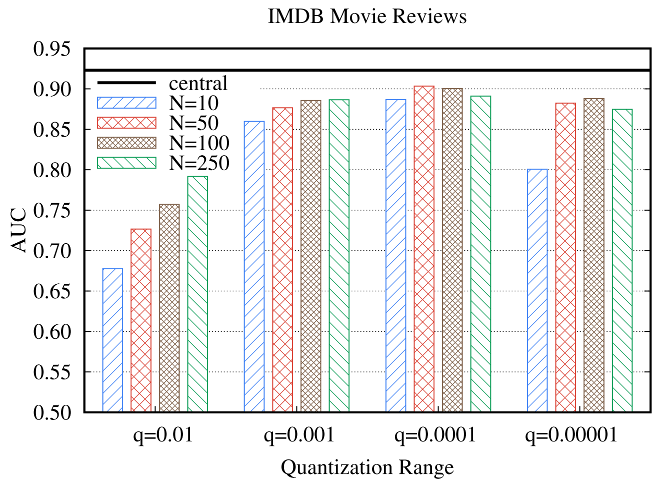
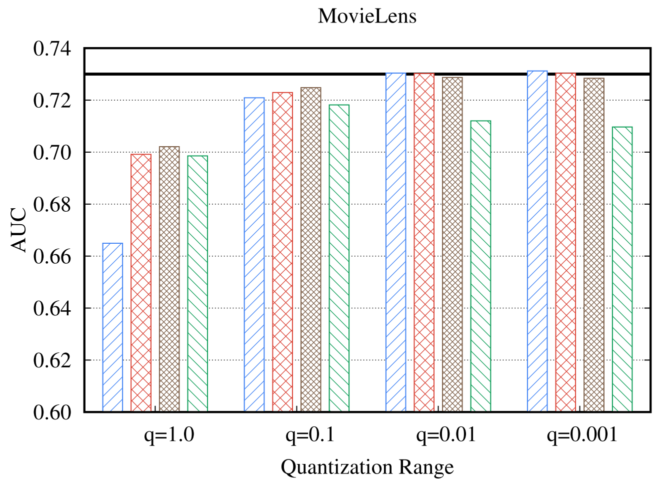



Fig. 3. Effects of quantization range (q) on the performance of the federated RR-LDP for different numbers of clients (N).

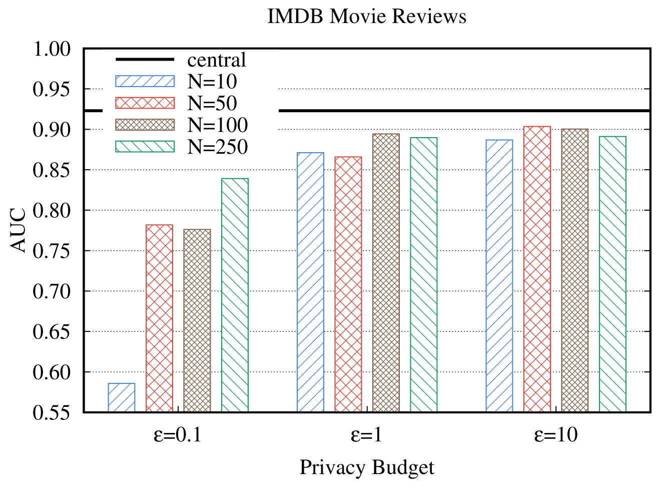
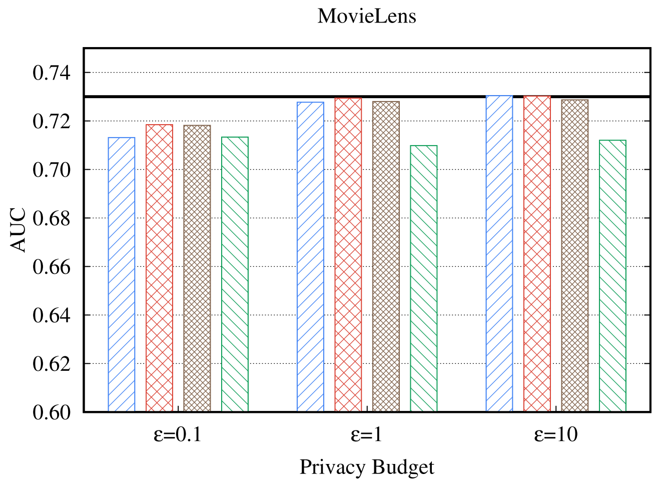



Fig. 4. Effects of privacy budget (ϵ) on the performance of the federated RR-LDP for different numbers of clients (N). See the legend of Fig. 3 for the bar colors.

proper q might be the subject of tuning experiments. However, given a feature set with values in the range [0, 1], a general rule of thumb for selecting a possibly good candidate for q would be to choose a value with the same magnitude as the learning rate parameter. In these experiments, we set the privacy parameter to 10, which corresponds to a very low perturbation ratio, and consequently, a very low privacy.

Furthermore, we also evaluate the second parameter of our privacy-preserving setting, i.e., the effect of different privacy budgets. We used a fixed quantization value in these experiments, which was equal to the local learning rate on the clients. Fig. 4 shows the performances for these settings, while the numerical results are listed in Tables 5 and 6. As expected, although the lower privacy budget values ensure better privacy, they result in inferior performance. However, as in the case of the quantization settings, the larger number of client devices can greatly improve the accuracy of the models. By involving more edge nodes, the variance can be reduced. Finally, the MovieLens 1M dataset, which uses mixed categorical and text features shows less sensitivity for varying the parameters than the IMDB database relying only on textual features.

In order to test the performance of our framework with more complex neural model structures, an additional LSTM layer with 50 units was also added to the above model. We trained the central baseline model with the same settings as above along with the FL simulations with $N = 100$ users, $R = 25$ communication rounds, $\epsilon = 1.0$ privacy budget as well as quantization parameters $q = 0.01$ (MovieLens) and $q = 0.001$ (IMDB Reviews).

Table 5
Effects of the privacy budget (ϵ) on the performance of the federated RR-LDP for different numbers of clients (N). All values are AUCs obtained for the IMDB Movie Reviews dataset.

	Number of clients			
ϵ	$N = 10$	$N = 50$	$N = 100$	$N = 250$
0.1	0.586	0.782	0.776	0.839
1.0	0.871	0.866	0.894	0.890
10.0	0.887	0.903	0.900	0.891
Central	*0.923*			

Table 6
Effects of the privacy budget (ϵ) on the performance of the federated RR-LDP for different numbers of clients (N). All values are AUCs obtained for the MovieLens 1M dataset.

	Number of clients			
ϵ	$N = 10$	$N = 50$	$N = 100$	$N = 250$
0.1	0.713	0.718	0.718	0.713
1.0	0.728	0.729	0.728	0.710
10.0	0.730	0.730	0.729	0.712
Central	*0.730*			

Table 7
Performance of LSTM in central as well as federated RR-LDP trainings. Federated simulations were run for $N = 100$, $R = 25$, $\epsilon = 1.0$, $q = 0.01$ (MovieLens), and $q = 0.001$ (IMDB).

	Central with LSTM		RR-LDP with LSTM	
Experiment	AUC	Accuracy	AUC	Accuracy
MovieLens 1M	0.745	0.682	0.756	0.676
IMDB Reviews	0.923	0.856	0.927	0.844

Table 7 shows the results of these experiments. It can be seen that introducing the LSTM layer slightly improves the performance of the centrally trained model for the MovieLens 1M dataset, however, it has no apparent effect in case of the IMDB Movie Reviews dataset. In our federated RR-LDP setting, a slight improvement can be seen due to the more complex model structure compared to the results obtained with the model without the LSTM layer (see Tables 1 and 2 for the corresponding AUC metrics). It can also be concluded from Table 7 that our federated RR-LDP approach works satisfactorily with more complex model structures as well, with only a slight, ∼1% loss in accuracy.

5.2. Text representation with string hashing

In this section, we demonstrate the usability of the hash-based text representation technique applied in our FL framework. For the IMDB Movie Reviews dataset, we use the term-frequency and word2vec embedding approaches as reference baselines, which are both specific to this dataset. In the case of the MovieLens 1M dataset, where training a specific embedding model is impractical, we utilize the pre-trained fastText model [18] to obtain reference embeddings for the keywords. FastText is a pre-trained embedding model using a subword-level representation approach often used in text classification tasks [54]. We applied various averaging schemes for the embedding vectors to obtain the final features used in training.

IMDB movie reviews. This dataset uses only natural language text for the semantic classification task, i.e., the classification is based solely on the features obtained with text hashing. Table 8 shows the performance of our hash-based representation in the binary semantic classification task with different hash bucket sizes. We compare these results to the reference embedding techniques using a linear (logistic regression, LR) and a Neural Network (NN) classifier. It can be seen that even for small bucket sizes,

Table 8
Performance measures on the IMDB Movie Reviews test set with the various embedding approaches and machine learning models.

	Logistic regression		Neural network model	
M (hash buckets)	Accuracy	AUC	Accuracy	AUC
5000	85.8	93.0	85.8	92.9
10000	87.0	94.0	87.1	93.8
20000	87.5	94.4	87.4	94.0
50000	88.0	94.9	87.9	94.4
90860[a]	88.1	95.0	88.1	94.6
References				
Term-frequency	88.1	94.8	87.7	94.0
Word2vec (train)[b]	86.3	93.5	84.2	91.6
Word2vec (full)[c]	87.5	94.4	85.1	91.4
Word2vec (limited)[d]	86.5	93.9	83.6	89.6

[a]90860 matches the dimension of the term-frequency embedding vector, i.e., the number of unique words in the corpus.
[b]The word2vec model is trained on the corpus of train data only, and uses a complete dictionary of all unique words with an embedding size of 300.
[c]The word2vec model is trained on the combined corpus of train and test data, and uses a complete dictionary of all unique words with an embedding size of 300.
[d]The vocabulary is limited to the top 5000 most frequent words. Out-of-vocabulary words are represented by zero vectors. A trainable embedding layer yields a dense representation for the word-level hashed inputs.

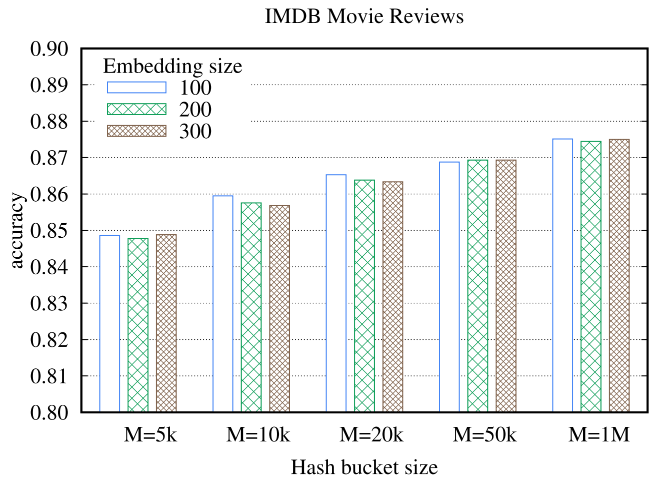
where hash collision is expected to be substantial, the results are promising. Compared to the count-based term-frequency embedding – where all unique words are considered – we lose roughly 2% both in accuracy and AUC with hashing. However, as expected, upon increasing the bucket size when computing the hash values, the performance gets better. With only 50k hash bins, which is still approximately half of all unique words, the reference performance could essentially be achieved for both applied models. When a bucket size that corresponds to the total number of unique words is applied in the hash function, i.e., we approximate a collision-free representation, the results are completely in line with the references. Interesting to note that the word2vec embeddings with a limited size vocabulary perform better with the linear classifier than the full-vocabulary embedding model. It is also worth noting that the word2vec model trained on the combined corpus of training and testing examples of the Movie Reviews dataset performs slightly better, in accordance with the expectations. The better performance is more apparent in the case of the LR classifier.

Using large sparse vectors, such as the term-frequency embedding or the hash representation with a large bucket size is not always practical, particularly on edge devices. Therefore, we also studied the possibility of dimension reduction for our hash-based representations by utilizing a trainable embedding layer which converts the raw hash vectors of text to a dense, real-valued vector. The same fully-connected neural network structure was used here as above, and only the format of the input stream and the position of the dropout layers were changed. The results are summarized in Table 9 and shown in Fig. 5. Performance scales only slightly but monotonically with the increasing size of hash buckets. The increasing embedding size, however, does not seem to have an apparent benefit on performance, which means that there is no advantage to using large embedding sizes in this particular use case. The utilization of the embedding layer introduces only a very small and practically negligible loss in accuracy. In turn, it provides a reduced dimension-dense representation of the hashed textual input.

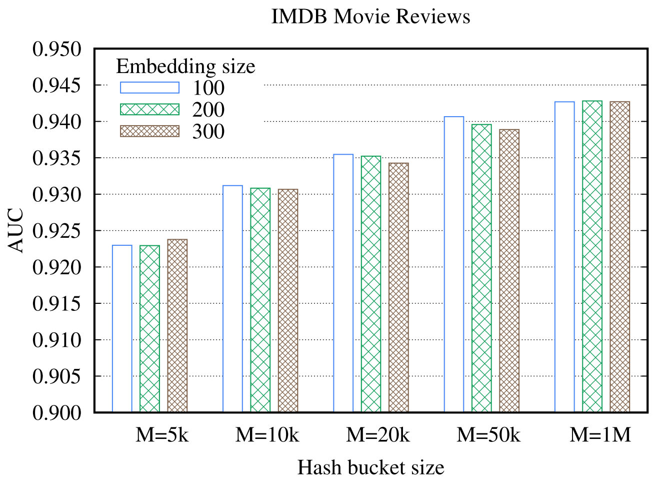
MovieLens 1M with additional keywords. We also evaluated the text hashing approach for the MovieLens 1M dataset, where the machine learning task is a classification-based ratings prediction. Table 10 summarizes the main results of this experiment, and shows accuracy and AUC values obtained for centrally

Table 9
Dimensionality reduction with a trainable embedding layer. Performance measures are obtained for the IMDB Movie Reviews sentiment analysis task.

	Embedding size					
	100		200		300	
M	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
5,000	84.9	92.3	84.8	92.3	84.9	92.4
10,000	85.9	93.1	85.8	93.1	85.7	93.1
20,000	86.5	93.5	86.4	93.5	86.3	93.4
50,000	86.9	94.1	86.9	94.0	86.9	93.9
1,000,000	87.5	94.3	87.4	94.3	87.5	94.3



(a) Accuracy



(b) AUC

Fig. 5. Scaling of performance metrics with increasing hash bucket and embedding sizes.

trained models using different embedding methods. Originally, the MovieLens 1M dataset contains numerical and categorical features. This feature set was further augmented in this study with a list of English keywords for movies downloaded from IMDB.[6] We first trained the same basic NN model used above for the movie reviews sentiment analysis task without the additional keywords and obtained an AUC of 67.9% as a baseline.

Table 10
Accuracy and AUC values obtained for the MovieLens 1M dataset with and without additional movie keywords with different embedding techniques. These results are obtained for centrally trained models without FL.

Text embedding	Accuracy	AUC
No text	64.3	67.9
fastText (100 flat)[a]	68.0	72.7
fastText (all averaged)[b]	68.2	73.2
fastText (100 averaged)[c]	68.1	73.2
Hash	68.1	72.9

[a]The 100 most common keyword embeddings were flattened to a 100 × 300 dimension embedding array and concatenated with the other features.
[b]All keyword embeddings for a given movie were averaged and a single 300 dimensional embedding vector was combined with the other features.
[c]The same as c, but only the 100 most common keywords per movie were considered in the averaging.

We then added the additional movie keywords to the dataset and trained the model again. The number of added keywords varies widely across the movies. The maximum number of keywords for a single movie is 1159, and there are movies without any keywords as well. The average number of keywords per movie is roughly 165, while the median value is 121. One keyword may occur multiple times on the keyword list for a given movie. The total number of unique keywords (total vocabulary size) after the preprocessing steps described above is 22,056.

As a reference to our hash embedding method, we used the pre-trained fastText model with different averaging strategies for the final embeddings. First, a single 300-dimension embedding vector was calculated for each movie by averaging the individual fastText embeddings for all keywords for the given movie. Second, the same averaging approach was used for the 100 most common keywords per movie. Finally, the individual embeddings of the 100 most common keywords were joined together to a (100 × 300) size embedding vector. In all cases, the resulting embedding vector was concatenated with the rest of the (normalized) numerical and categorical features in the original MovieLens dataset.

For the hash representation, we used the sentence-level hash embedding as described above. In particular, the individual bucket-size one-hot embedding vectors of the keywords were combined to obtain a single, bucket-size dimensional binary vector with 1s in positions determined by the hash values of the keywords. The resulting embedding vector was then concatenated with the rest of the normalized features in the original dataset.

As it can be seen in Table 10, the performance of the model can considerably be improved by adding the keywords. With this augmentation of the feature set, the AUC can be increased by at least 5%. However, the performance is not significantly affected by the chosen embedding approach. With the largest feature space, where (100 × 300) text embedding size was used, the AUC of the rating prediction is 72.7%. The averaged fastText embeddings are slightly better with an AUC of 73.2%. This is apparently independent of the averaging strategy, i.e., the very same AUC can be obtained if we average over all keywords or the top 100 most common ones. For the hash representation, we obtained an AUC of 72.9%, which is just slightly below the averaged fastText result but a little better than the large dimension fastText experiment. In a private FL setting with 100 clients and a privacy budget of $\epsilon = 1.0$, the corresponding AUC with the hash-based representation is 72.8% (see Table 6), which is almost identical to the AUC of the centrally trained model without any privatization effort.

6 https://www.imdb.com/

6. Conclusions

In this study, we proposed a privacy-preserving FL framework that allows us to train machine learning models privately with local data stored on a set of client devices. In order to ensure enhanced client privacy, a randomized-response-based locally differentially private technique is utilized for the updates of the local model. In particular, the differences between the weights of the client and central server models are binary quantized and randomized before they are sent back for the server-side aggregation. We mainly focused on NLP-related examples in this study, where the client data is natural language text, however, our proposed framework was designed to be functional with all feature types. For textual feature extraction on the client-side, a low-cost and vocabulary-free text embedding approach is applied based on the polynomial rolling hash algorithm. Due to the one-way only conversion of the hash function that makes the inversion computationally difficult, hashing the local textual input directly on the user devices also improves the efficiency of the privacy preservation in our framework.

We showed that our proposed private federated learning approach causes only a slight decrease in accuracy in a few example classification tasks involving textual features. There are several benefits to each component in our framework. Namely, the hash representation allows low storage requirements in text embedding, small computational complexity with no need for any sophisticated language model, and easy textual feature inclusion into the proposed private federated learning method. Concerning our privacy-preserving method, the randomized response algorithm ensures local differential privacy, and eases the communication burden with low-cost bitwise vectors during the data transfer phases from the clients to the server. We evaluated our approach on two datasets, including a binary sentiment classification task using textual features only, and a movie rating prediction that combines categorical features with textual ones.

We found that our proposed locally differentially private federated learning framework with hashed text representation performs in line with other more sophisticated embedding approaches and centrally trained machine learning models without any endeavors to preserve user privacy.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The research presented in this paper, carried out by the University of Szeged, was supported by the Ministry of Innovation and Technology, Hungary and the National Research, Development and Innovation Office, Hungary within the framework of the Artificial Intelligence National Laboratory Programme, and by project TKP2021-NVA-09, implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: A. Singh, J. Zhu (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, in: Proc. Mach. Lear. Res., vol. 54, PMLR, Fort Lauderdale, FL, USA, 2017, pp. 1273–1282.

[2] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) 1–19, http://dx.doi.org/10.1145/3298981.

[3] Q. Xia, W. Ye, Z. Tao, J. Wu, Q. Li, A survey of federated learning for edge computing: Research problems and solutions, High-Confi. Comput. 1 (1) (2021) 100008, http://dx.doi.org/10.1016/j.hcc.2021.100008.

[4] M. Aledhari, R. Razzak, R.M. Parizi, F. Saeed, Federated learning: A survey on enabling technologies, protocols, and applications, IEEE Access 8 (2020) 140699–140725, http://dx.doi.org/10.1109/ACCESS.2020.3013541.

[5] Z. Li, Z. Huang, C. Chen, C. Hong, Quantification of the leakage in federated learning, 2020, arXiv:1910.05467.

[6] A. Wainakh, F. Ventola, T.M. ig, J. Keim, C.G. Cordero, E. Zimmer, T. Grube, K. Kersting, M. Mühlhäuser, User label leakage from gradients in federated learning, 2021, arXiv:2105.09369.

[7] S.P. Kasiviswanathan, H.K. Lee, K. Nissim, S. Raskhodnikova, A. Smith, What can we learn privately? SIAM J. Comput. 40 (3) (2011) 793–826, http://dx.doi.org/10.1137/090756090.

[8] X. Xiong, S. Liu, D. Li, Z. Cai, X. Niu, A comprehensive survey on local differential privacy, in: A.M. Del Rey (Ed.), Secur. Commun. Netw. 2020 (2020) 1–29, http://dx.doi.org/10.1155/2020/8829523.

[9] L. Sun, J. Qian, X. Chen, P.S. Yu, LDP-FL: Practical private aggregation in federated learning with local differential privacy, 2020, arXiv:2007.15789.

[10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186, http://dx.doi.org/10.18653/v1/N19-1423.

[11] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[12] Ú. Erlingsson, V. Pihur, A. Korolova, Rappor: Randomized aggregatable privacy-preserving ordinal response, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014, pp. 1054–1067, http://dx.doi.org/10.1145/2660267.2660348.

[13] A.G. Thakurta, A.H. Vyrros, U.S. Vaishampayan, G. Kapoor, J. Freudiger, V.R. Sridhar, D. Davidson, Learning new words, 2017, US Patent 9, 594, 741.

[14] O. Feyisetan, B. Balle, T. Drake, T. Diethe, Privacy- and utility-preserving textual analysis via calibrated multivariate perturbations, in: Proceedings of the 13th International Conference on Web Search and Data Mining, WSDM '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 178–186, http://dx.doi.org/10.1145/3336191.3371856.

[15] O. Feyisetan, T. Diethe, T. Drake, Leveraging hierarchical representations for preserving privacy and utility in text, in: 2019 IEEE International Conference on Data Mining, ICDM, IEEE, 2019, pp. 210–219, http://dx.doi.org/10.1109/ICDM.2019.00031.

[16] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, J. Attenberg, Feature hashing for large scale multitask learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, Association for Computing Machinery, New York, NY, USA, 2009, pp. 1113–1120, http://dx.doi.org/10.1145/1553374.1553516.

[17] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.

[18] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Trans. Assoc. Comput. Linguist. 5 (2017) 135–146, http://dx.doi.org/10.1162/tacl_a_00051.

[19] C. Dwork, A firm foundation for private data analysis, Commun. ACM 54 (1) (2011) 86–95, http://dx.doi.org/10.1145/1866739.1866758.

[20] K. Fukuchi, Q.K. Tran, J. Sakuma, Differentially private empirical risk minimization with input perturbation, in: International Conference on Discovery Science, Springer, 2017, pp. 82–90, http://dx.doi.org/10.1007/978-3-319-67786-6_6.

[21] Y. Kang, Y. Liu, B. Niu, X. Tong, L. Zhang, W. Wang, Input perturbation: A new paradigm between central and local differential privacy, 2020, arXiv:2002.08570.

[22] R. Bassily, A. Smith, A. Thakurta, Private empirical risk minimization: Efficient algorithms and tight error bounds, in: 2014 IEEE 55th Annual Symposium on Foundations of Computer Science, 2014, pp. 464–473, http://dx.doi.org/10.1109/FOCS.2014.56.

[23] I. Hegedűs, M. Jelasity, Distributed differentially private stochastic gradient descent: An empirical study, in: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP, IEEE Computer Society, 2016, pp. 566–573, http://dx.doi.org/10.1109/PDP.2016.19.

[24] K. Chaudhuri, C. Monteleoni, Privacy-preserving logistic regression, in: Proceedings of the 21st International Conference on Neural Information Processing Systems, NIPS '08, Curran Associates Inc., Red Hook, NY, USA, 2008, pp. 289–296, http://dx.doi.org/10.5555/2981780.2981817.

[25] K. Chaudhuri, C. Monteleoni, A.D. Sarwate, Differentially private empirical risk minimization, J. Mach. Learn. Res. 12 (2011) 1069–1109, arXiv:0912.0071.

[26] T. Wang, X. Zhang, J. Feng, X. Yang, A comprehensive survey on local differential privacy toward data statistics and analysis, Sensors 20 (24) (2020) 7030, http://dx.doi.org/10.3390/s20247030.

[27] S.L. Warner, Randomized response: A survey technique for eliminating evasive answer bias, J. Amer. Statist. Assoc. 60 (309) (1965) 63–69, http://dx.doi.org/10.1080/01621459.1965.10480775.

[28] C. Dwork, A. Roth, The algorithmic foundations of differential privacy, Found. Trends Theor. Comput. Sci. 9 (3–4) (2014) 211–407, http://dx.doi.org/10.1561/0400000042.

[29] Y. Wang, X. Wu, D. Hu, Using randomized response for differential privacy preserving data collection, in: EDBT/ICDT Workshops, vol. 1558, 2016, pp. 0090–6778.

[30] W. Zhang, X. Li, Data privacy preserving federated transfer learning in machinery fault diagnostics using prior distributions, Struct. Health Monit. 21 (4) (2022) 1329–1344, http://dx.doi.org/10.1177/14759217211029201.

[31] W. Zhang, X. Li, Federated transfer learning for intelligent fault diagnostics using deep adversarial networks with data privacy, IEEE/ASME Trans. Mechatronics 27 (1) (2022) 430–439, http://dx.doi.org/10.1109/TMECH.2021.3065522.

[32] V. Mothukuri, R.M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, G. Srivastava, A survey on security and privacy of federated learning, Future Gener. Comput. Syst. 115 (2021) 619–640, http://dx.doi.org/10.1016/j.future.2020.10.007.

[33] N. Truong, K. Sun, S. Wang, F. Guitton, Y. Guo, Privacy preservation in federated learning: An insightful survey from the GDPR perspective, 2021, arXiv:2011.05411.

[34] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, 2014, arXiv:1412.6115.

[35] D. Barth-Jones, The 're-identification' of governor william weld's medical information: A critical re-examination of health data identification risks and privacy protections, Then now (2012) http://dx.doi.org/10.2139/ssrn.2076397.

[36] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: Empirical Methods in Natural Language Processing, EMNLP, Association for Computational Linguistics, 2014, pp. 1532–1543, http://dx.doi.org/10.3115/v1/D14-1162.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L.u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017, pp. 5998–6008.

[38] M. Gupta, P. Agrawal, Compression of deep learning models for text: A survey, ACM Trans. Knowl. Discov. Data 16 (4) (2022) http://dx.doi.org/10.1145/3487045.

[39] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, J. Attenberg, Feature hashing for large scale multitask learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, Association for Computing Machinery, New York, NY, USA, 2009, pp. 1113–1120, http://dx.doi.org/10.1145/1553374.1553516.

[40] C. Zhang, Y. Liu, Y. Xie, S.I. Ktena, A. Tejani, A. Gupta, P.K. Myana, D. Dilipkumar, S. Paul, I. Ihara, P. Upadhyaya, F. Huszar, W. Shi, Model size reduction using frequency based double hashing for recommender systems, in: Fourteenth ACM Conference on Recommender Systems, RecSys '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 521–526, http://dx.doi.org/10.1145/3383313.3412227.

[41] J. Serrà, A. Karatzoglou, Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks, in: Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 279–287, http://dx.doi.org/10.1145/3109859.3109876.

[42] D. Tito Svenstrup, J. Hansen, O. Winther, Hash embeddings for efficient word representations, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017, pp. 4935–4943.

[43] L. Chi, B. Li, X. Zhu, Context-preserving hashing for fast text classification, in: Proceedings of the 2014 SIAM International Conference on Data Mining, SDM, pp. 100–108, http://dx.doi.org/10.1137/1.9781611973440.12.

[44] L. Argerich, J.T. Zaffaroni, M.J. Cano, Hash2Vec, feature hashing for word embeddings, 2016, arXiv:1608.08940.

[45] W.-C. Kang, D.Z. Cheng, T. Yao, X. Yi, T. Chen, L. Hong, E.H. Chi, Deep hash embedding for large-vocab categorical feature representations, 2020, arXiv Preprint abs/2010.10784, arXiv:2010.10784.

[46] R.M. Karp, M.O. Rabin, Efficient randomized pattern-matching algorithms, IBM J. Res. Dev. 31 (2) (1987) 249–260, http://dx.doi.org/10.1147/rd.312.0249.

[47] A.L. Maas, R.E. Daly, P.T. Pham, D. Huang, A.Y. Ng, C. Potts, Learning word vectors for sentiment analysis, in: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Portland, Oregon, USA, 2011, pp. 142–150.

[48] F.M. Harper, J.A. Konstan, The MovieLens datasets: History and context, ACM Trans. Interact. Intell. Syst. 5 (4) (2015) http://dx.doi.org/10.1145/2827872.

[49] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2016, Software available from tensorflow.org. arXiv:1603.04467.

[50] S. Bird, E. Klein, E. Loper, Natural Language Processing with Python, first ed., O'Reilly Media, Inc., 2009.

[51] R. Řehůřek, P. Sojka, Software framework for topic modelling with large corpora, in: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, ELRA, Valletta, Malta, 2010, pp. 45–50.

[52] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, 2015, arXiv:1412.6980.

[53] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780, http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[54] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of tricks for efficient text classification, in: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, vol. 2, Association for Computational Linguistics, 2017, http://dx.doi.org/10.18653/v1/e17-2068.