Training a Tokenizer for Free with Private Federated Learning

Anonymous ACL submission

Abstract

Federated learning with differential privacy, i.e. private federated learning (PFL), makes it possible to train models on private data distributed across users' devices without harming privacy. PFL is efficient for models, such as neural networks, that have a fixed number of parameters, and thus a fixed-dimensional gradient vector. Such models include neural-net language models, but not tokenizers, the topic of this work. Training a tokenizer requires frequencies of words from an unlimited vocabulary, and existing methods for finding an unlimited vocabulary need a separate privacy budget.

A workaround is to train the tokenizer on publicly available data. However, in this paper we first show that a tokenizer trained on mismatched data results in worse model performance compared to a privacy-violating "oracle" tokenizer that accesses user data, with perplexity increasing by 20 %. We also show that sub-word tokenizers are better suited to the federated context than word-level ones, since they can encode new words, though with more tokens per word.

Second, we propose a novel method to obtain a tokenizer without using any additional privacy budget. During private federated learning of the language model, we sample from the model, train a new tokenizer on the sampled sequences, and update the model embeddings. We then continue private federated learning, and obtain performance within 1 % of the "oracle" tokenizer. Since this process trains the tokenizer only indirectly on private data, we can use the "postprocessing guarantee" of differential privacy and thus use no additional privacy budget.

1 Introduction

Learning a language model (LM) requires text data that in many situations is private, resides on people's devices, and should stay there. In federated learning (McMahan et al., 2017), a central server learns a model by receiving statistics, like parameter updates, from many devices. Though devices send only statistics and not the raw data, federated learning by itself can leak information about the data (Shokri et al., 2017; Song et al., 2017). Private federated learning (PFL) (McMahan et al., 2018; Geyer et al., 2017) uses differential privacy (Dwork et al., 2006, 2014) to mitigate the privacy leaks by limiting the user's impact on the final model.

It is known how to train neural-net language models using PFL (McMahan et al., 2018). However, an important part of language modeling is tokenization: turning a text into a sequence of symbols from a fixed-size symbol set. To obtain a tokenizer, published research on private federated learning of language models uses either of two approaches, neither of which are satisfactory. One approach is to train the tokenizer on user data directly. The commonly-used LEAF dataset (Caldas et al., 2018) and works relying on it (Li et al., 2021; Hu et al., 2021; Yu et al., 2020) assume access to the training data to create the tokenizer. This is not relevant to real-world use cases and undermines user privacy. The other approach is to use public data to obtain the tokenizer (McMahan et al., 2018). This is sensible from a privacy perspective, but as we show the resulting distribution mismatch harms performance, resulting in 10%-20% drop compared to using an "oracle" tokenizer trained directly on users' private data.

There are two common types of tokenization, which are affected by mismatched distributions in different ways: word and sub-word tokenization. Figure 1 illustrates these. A word-level tokenizer produces a symbol for each word, and assigns an out-of-vocabulary token (OOV) to any unseen word. Text from mismatched distributions will generally contain unseen words, which means the correct word cannot be predicted, and the context becomes less meaningful when predicting the

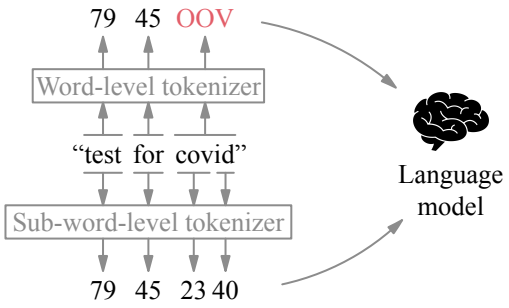


Figure 1: Word-level and sub-word-level tokenization. A word-level tokenizer can generate an "out-of-vocabulary" (OOV) symbol, which it is hard for a language model to use.

next word. Sub-word tokenization, on the other hand, splits some words into multiple smaller tokens. This type of tokenization is generally chosen to minimize the average number of tokens per word on training data. Current centrally trained models use sub-word tokenization such as Byte-Pair Encoding (Sennrich et al., 2016), SentencePiece (Kudo and Richardson, 2018), or WordPieces (Schuster and Nakajima, 2012). Nevertheless, mismatched tokenizations in sub-word methods cause an increase in the number of tokens per word, and thus decrease the amount of context the model can use to predict the distribution of the next word.

In this work we present a general framework to approach training language models in private federated learning by including tokenization as part of the training pipeline. Our contributions are: (1) we uncover the performance gaps when the models use the tokenizer obtained from a different distribution vs the tokenizer obtained from the underlying distribution. For word-level tokenization we show that a tokenizer trained on public data reduces the next-word prediction accuracy of 10–20 % compared to a tokenizer estimated on user data. (2) We demonstrate significant benefits of switching tokenizers from word to sub-word level, thus eliminating the out-of-vocabulary problem. (3) We propose a new method that samples data from an existing model, e.g. from the prior PFL run, and uses that data to initialize a new tokenizer. Our approach can update the tokenizer between iterations of the same PFL run by modifying model embeddings with new tokenizations and significantly boosting performance. Crucially, since the language model is trained with differential privacy, the "postprocessing guarantee" of differential privacy means that training the tokenizer with our approach does not use any additional privacy budget.

2 Private federated learning

Machine-learned models work best if they are trained on the correct distribution of the data, in this paper text data. In many scenarios text data is private and contained on people's devices, and should stay there. To train a global model without harming privacy, we use federated learning (McMahan et al., 2017) with differential privacy (Dwork et al., 2006, 2014).

Federated learning involves devices sending not the data, but statistics, e.g. model gradients, computed on that data. To train neural networks, the standard algorithm is *federated averaging* (McMahan et al., 2017). At each iteration t, the server randomly selects a subset of m participants S_m and distributes the current global model M^t. Each participant takes a number of gradient steps to train on their private data and submits the sum G_i^t of the gradients to the server. The server takes a step (with step size η) in the direction of the average gradient to create the new global model:

$$M^{t+1} = M^t + \frac{\eta}{m} \sum_{i=1}^{m} G_i^t \qquad (1)$$

2.1 Federated Learning with Differential Privacy

The global model M^{t+1} might still reveal private information including user participation in training (Shokri et al., 2017; Song et al., 2017; Melis et al., 2019). To mitigate this threat, we can combine federated learning with differential privacy (DP) (Dwork et al., 2006, 2014), to give *private federate learning* (McMahan et al., 2018). Differential privacy gives a strong guarantee: it limits the advantage that a computationally unconstrained adversary has in inferring whether an individual's data is contained in the data set that the statistics are computed from. (ϵ, δ)-differential privacy parametrizes this advantage by ϵ (the maximum privacy loss) and δ (a slack term). The common mechanism to provide differential privacy in a federated learning setting is the Gaussian mechanism that uses the *moments accountant* (Abadi et al., 2016). For each participant, the model parameters are *clipped* to a norm S, i.e., multiplied by $\min(1, S/\|G^t\|_2)$, to bound the sum's sensitivity to any individual's data. Second, Gaussian noise $\mathcal{N}(0, \sigma^2)$ is added to the final sum. How much privacy budget is spent

depends on the variance σ^2 relative to the magnitude of individual updates, the total population, the number of contributions in each iteration, and the total number of iterations (for more details, see McMahan et al., 2018; Balle et al., 2018).

2.2 Privately finding vocabulary items

Central differential privacy with the Gaussian mechanism and the moments accountant is efficient in terms of utility vs privacy loss, but it does come with restrictions. The sum of individual contributions, which the noise is added to, must be of finite and fixed size. This is not a problem for training neural networks. However, training a tokenizer requires frequencies for an exponential-size set of sequences, as does training a traditional N-gram model. Differentially private algorithms to compute histograms over sets of elements (e.g. words) distributed over devices are called "heavy hitters" algorithms (Bassily et al., 2017; Zhu et al., 2020; Apple, 2017). These algorithms require a separate and large privacy budget. In section 5 we will compare with a heavy hitters algorithm.

Another way of finding vocabulary items privately is to train a neural-net generative model. Beaufays et al. (2019) trains a separate, character-level LSTM model to generate the new words. However, the proposed method is only shown to work for discover OOVs in a word-level model and also requires separate training and a privacy budget.

3 Tokenization in Language Modeling

A language model is a model that assigns probabilities to sequences of tokens. In this paper, it is always an autoregressive model with parameters θ: $P_\theta(s) = P_\theta(t_2|t_1 = \texttt{BOS}) \cdot P_\theta(t_3|t_1 = \texttt{BOS}, t_2) \cdots P_\theta(t_n = \texttt{EOS}|t_1 = \texttt{BOS}, \ldots, t_{n-1})$, where each term in this equation is normalized over all possible values of the current token. Local normalization is useful when decoding input, like in speech recognition or a keyboard (Hard et al., 2018). For this paper, we assume that a corpus is segmented into sentences. A tokenizer τ then converts each sentence s in the dataset into a sequence of n tokens $\tau(s) = [\texttt{BOS}, t_2, .., t_{n-1}, \texttt{EOS}]$, which is fed into the language model. There are two types of tokenization, highlighted in Figure 1: word-level and sub-word-level. Using a sub-word tokenizer will be key to the algorithm this paper proposes.

The next section will discuss the two types of tokenizers and their consequences for out-of-vocabulary tokens and the performance of language models based in them. Section 3.2 will discuss the complex topic of how to compare performance across different tokenizations.

3.1 Word-level vs sub-word-level tokenization

The type of tokenization that papers about language models in federated learning commonly use is word-level tokenization (McMahan et al., 2017). For a vocabulary of size N the tokenizer assigns a unique token for top-N most popular words in the dataset while other words receive an out-of-vocabulary token OOV, as highlighted in Figure 1. Some papers (e.g. McMahan et al., 2018) build the tokenizer from a publicly available dataset, others including the LEAF benchmark (Caldas et al., 2018) build the tokenizer from users' training data. OOV tokens in the word history make it harder for a language model to predict the next word.

The other type of tokenization is sub-word tokenization, for which there are two popular schemes: byte-pair encoding (BPE) (Sennrich et al., 2016) and WordPieces (Schuster and Nakajima, 2012). We focus on BPE which unlike WordPieces guarantees the absence of OOVs as there exists a token for every byte. However, the number of tokens required to encode each word can change significantly depending on the dataset that the tokenizer was trained on. As highlighted in Figure 1, a tokenizer trained on data from before the COVID-19 pandemic would generate multiple tokens for the word "covid".

Generating longer token sequences makes it harder for the language model to keep track of the context, degrading its performance. Even LSTMs and transformers, which in theory can use arbitrarily long history, have imperfect memory.

3.2 Evaluating language models across tokenizations

Comparing language models across tokenizations is a complex problem. For example, when comparing word-level language models using perplexity, often OOVs are ignored which gives an edge to the language model with more OOVs, which is the opposite of what is desired. The following sections detail the problems when comparing sub-word language models.

3.2.1 Comparing word-level with sub-word

Since a word-level language model has a closed vocabulary, it outputs probabilities only on in-vocabulary words, artificially lowering the perplexity of closed-vocabulary LMs, particularly on data with a large number of OOVs. Removing those same words in evaluating a sub-word language model, would disadvantage it.

A better alternative, which this paper will use, is to compare model performance the word-level accuracy. The most accurate way would be to find the word with the highest probability by summing over sequences of tokens. However, we choose a simpler, though less accurate method (similar to Likhomanenko et al., 2019): repeatedly generate the best tokens within each word's bounds and only accept the word as accurate if all generated tokens were correct.

3.2.2 Comparing sub-word with sub-word

It is possible to meaningfully compare perplexities of two language models with different sub-word tokenizations (Mielke, 2019). Though the language model assigns probability mass to all token sequences, a single sentence can have multiple corresponding token sequences, only one of which will be chosen by the tokenizer. Some of the probability mass will therefore be lost to never-occurring token sequences. However, it is unfeasible to sum over all token sequences (Likhomanenko et al., 2019).

The danger with comparing perplexities directly is that since models with different tokenizers operate on different sets of tokens the number of tokens needed to encode each sentence is different in general (Mielke, 2019). Nevertheless, note that all models assign a probability to a sentence (with the approximation above). To compute the perplexity in such a way that it can be compared across tokenizers, use the same denominator in computing the perplexity: the number of words in the sentence instead of number of tokens, which depends on the tokenizer. Therefore we define the perplexity as:

$$ppl_{\theta,\tau}(s) = \exp\left(\frac{-\log(P_{\theta,\tau}(s))}{\|s\|_w}\right) \qquad (2)$$

where $\|s\|_w$ counts the number of words in the sentence s. To generalize from a single sentence to a dataset, replace s with the concatenation of all sentences in the dataset.

4 Learning a Tokenizer with Private Federated Learning

Problem definition. We aim to obtain a tokenizer that works well on users' federated data without compromising user privacy. First, we aim to find the appropriate tokenization scheme, and second, given the tokenization scheme obtain the right approximation of user data to train the tokenizer.

Setting We focus on a common application of federated learning: training a language model, parameterized by θ, using federated learning with differential privacy. In our setting each user u_i has a dataset d_i of private texts from a private distribution of user data $\mathcal{D}$. The trained model will be evaluated against a held-out dataset $\mathcal{D}_{test}$, e.g. a mix of all user data, which in practice must be replaced by federated evaluation.

We assume that the central server does not have access to the user data distribution $\mathcal{D}$ and can only approximate it with the publicly available dataset $\mathcal{D}_{pub}$. We assume the public data is some commonly available dataset, such as Wikipedia (Merity et al., 2017). The tokenizer trained on this public data will be τ_{pub}. For comparison we assume the existence of an *oracle* tokenizer τ_o initialized on users' training data $\mathcal{D}$.

Papers that study language models in federated learning commonly use word-level tokenization. While some papers (e.g. McMahan et al., 2018), build the vocabulary using publicly available dataset, others (e.g. Yu et al., 2020; Caldas et al., 2018) explicitly use the federated training data, even though in real-world scenarios the analogous data would be unavailable and it violates privacy guarantees when used in PFL (Li et al., 2021).

4.1 Sampling from a PFL-trained language model

To address the problem of learning a good tokenizer we first propose to use a sub-word tokenizer with an open vocabulary. This allows the language model trained with such a tokenizer to represent any word, if inefficiently. It is then possible to query the language model to find new words as the model can utilize this open vocabulary. This is the core of the Algorithm 1 that this paper introduces.

Figure 2 shows the proposed pipeline. A language model is trained with private federated learning. This results (on the left) in a model matched

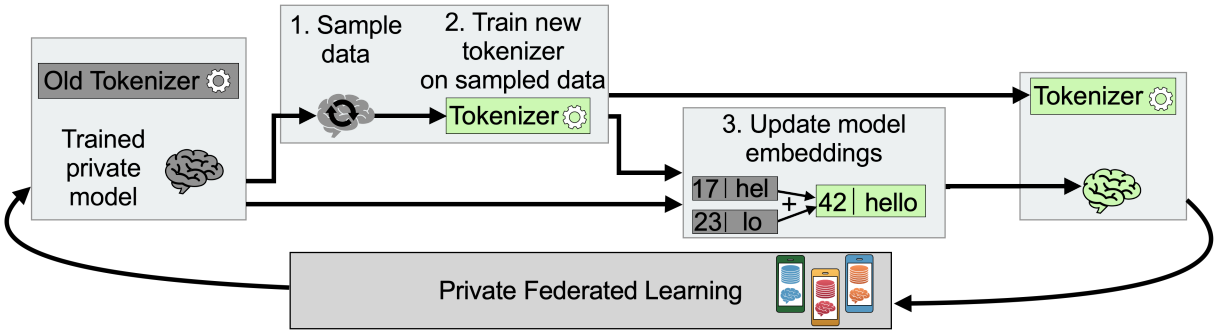


Figure 2: New pipeline for updating the tokenizer through model sampling.

with an old, stale tokenizer. The next block queries the language model to produce a better tokenizer, with a method that section 4.2 will detail. The block after that updates the language model for the new tokenizer, using reasonable guesses for the new parameters. This results in a new LM-tokenizer combination that can be trained further with PFL.

We assume that the language model obtained with the stale tokenizer is trained with a certain privacy budget. The postprocessing guarantee of differential privacy (Dwork, 2011) means that the steps other than private federated learning do not consume any further budget. The function UPDATE in Algorithm 1 performs the on-server steps. The following sections will give more detail.

4.2 New tokenizer from a trained LM

Training a tokenizer requires text data. Since the raw data is not available, we propose to instead sample from the LM matched with the stale tokenizer, as detailed in Algorithm 1. The SAMPLETOKENS function samples from the language model, drawing sequences of tokens according to the probabilities that the model assigns to them. The SAMPLE function then converts these sequences in the old tokenization into word sequences, by decoding with τ_{pub}. Once a large enough corpus of word-level sentences has been produced, training a tokenizer proceeds as normally (the TRAINTOKENIZER function is not specified).

4.3 Adapting the language model to the new tokenizer

After a new tokenizer τ has been trained, the language model, trained with τ_{pub}, must be updated to work with the new tokenizer. Neural-net language models use an embedding layer to convert the provided tokens into multi-dimensional vectors. It is the embedding vectors that are most important to modify when changing the tokenization. The rest of the model only consumes the embedding vector. It is not possible to find the optimal parameters without further training of both embeddings and other layers, but we propose an algorithm to find a reasonable starting point, in the function REMAP(τ, τ_{pub}) in Algorithm 1.

REMAP iterates over the tokens from the new tokenizer τ and creates the mapping from the tokens' embedding in the public tokenizer τ_{pub} to the new token's embedding. In some cases it is a one-to-one mapping, but when the new token accumulates multiple tokens in τ_{pub} we split the weight equally between each token.

Once we have the mapping map we modify the embedding layer of the model by performing matrix multiplication, i.e. $\theta.\text{embedding} = map \cdot \theta.\text{embedding}$. The resulting model can accept the tokens from the new tokenizer τ, and can participate in future training in federated learning.

5 Experiments

We evaluate our approach by first looking at performance of tokenizers trained on the distributions matched and mismatched to real data, we then test the proposed federated sampling on different datasets for federated learning.

5.1 Experimental setup.

We use two datasets common in the federated learning literature (Kairouz et al., 2019). While both use English, there is nothing about our experiments that is specific to this language, and multilingual datasets can further benefit from using SentencePiece tokenization (Kudo and Richardson, 2018),.

- Reddit data – this dataset is taken from the LEAF benchmark (Caldas et al., 2018) and contains over a million users that have multiple posts on the Reddit platform. As proposed

Algorithm 1 Model sampling algorithm

Inputs: model θ, current sentence s, new tokenizer τ, public tokenizer τ_{pub}, size of the sampled dataset corpus_size.

function SAMPLETOKENS(θ, s)
  $t_{next} \sim_\theta t_k|s$
  if $t_{next} = \texttt{EOS}$ **then**
    return $s +\!\!+ t_{next}$
  else
    return SAMPLETOKENS($\theta, s +\!\!+ t_{next}$)

function SAMPLE(θ, τ)
  return τ.decode(
    SAMPLETOKENS(θ, [BOS]))

function REMAP(τ_{pub}, τ)
  map = zeros(τ.size, τ_{pub}.size)
  for token, tid $\leftarrow \tau$.vocab **do**
    tokens = τ_{pub}.decode(token)
    for token $\leftarrow$ tokens **do**
      $\text{tid}_{pub} = \tau_{pub}$.vocab[token]
      map[tid_{pub}, tid] = 1/len(tokens)
  return map

function UPDATE(θ, τ_{pub})
  while len(corpus) $<$ corpus_size **do**
    corpus $\leftarrow$ SAMPLE($\theta, \emptyset, l_{max}$)
  τ = TRAINTOKENIZER(corpus)
  map = REMAP(τ_{pub}, τ)
  θ.embedding = map $\cdot$ θ.embedding
  return θ, τ

by LEAF, we limit each user to contain at most 1600 tokens and use 10 % of users for faster training.

- StackOverflow data – this data is taken from Kaggle (Kaggle, 2021) and processed with the TensorFlow Federated framework. The train split of the dataset contains 342k users and we select at most 1600 tokens per user.

Model parameters. We use an LSTM model with 3 layers, and total parameters of 14M. We also use a Transformer language model (Vaswani et al., 2017) with 6 layers and the same total number of parameters as the LSTM (see Appendix A). Each model is trained from scratch.

Hyper-parameters. We set the privacy budget to $\epsilon = 2$ and $\delta = 10^{-6}$ – a common privacy regime (Kairouz et al., 2019). For the "heavy hitters" baseline we use local DP with an additional privacy budget of $\epsilon = 8$.[1] The overall population for the moments accountant is assumed to be 10m. We use a cohort size of $20,000$ for each round and train all models for $5,000$ iterations. We use Adam (Kingma and Ba, 2015) for central optimization with learning rate set to 0.5. For the clients we use SGD and train for 1 local epoch with batch size set to 16 and local learning rate set to 0.1, and an L_2 clipping bound for DP of 0.5.

Vocabulary size. We assume that the tokenizer has a moderate vocabulary size such as 10,000 tokens (we experiment with larger vocabularies in Appendix A). Smaller vocabularies reduce model size and, therefore, might be better for deployment on devices and communication with the global server.

Tokenizer details. To train an initial tokenizer we use a popular and public Wikipedia dataset (Merity et al., 2017). It may seem like the distribution of Wikipedia data is artificially far from the distributions of Reddit and StackOverflow data. However, the server might not have the right prior possibly due to a natural *distribution shift* (Miller et al., 2020) of typed texts (such as an emerging topic of which there were plenty recently).

We use BPE and WordLevel tokenization algorithms from the HuggingFace Tokenizer library (Huggingface, 2021). Each user post is surrounded by special tokens BOS and EOS. We also tried WordPieces tokenization which has slightly better performance than BPE but cannot encode all words and is therefore less applicable in FL.

Note on splitting data. Whereas the original LEAF dataset for Reddit proposes to split each user's data we argue that in real life not every user might have a chance to participate in the training. Therefore, we split users into two distinct training and test sets and evaluate the model on data from the users who have never participated in the training. This results in notably increased test perplexity but provides a clear separation between training and inference modes.

5.2 Comparing tokenization schemes

Table 1 summarizes experiments that use different tokenization schemes. We compute statistics on tokenizers: the average share of OOV tokens for the word-level scheme and the average number of tokens required to encode one word for the sub-word scheme. To compare the effect of each tokenizer

[1] Budgets for local and central privacy are not immediately comparable, but see Feldman et al. (2021).

Table 1: Word accuracy suffers for word-level tokenization that uses mismatched data.

Type	Data to train τ	τ statistics OOV (%)	τ statistics Tokens per word	Word Accuracy (%)
Reddit				
Word-Level	Wiki	13.0	1.00	17.7
Word-Level	Oracle	5.5	1.00	24.1
BPE	Wiki	0.0	1.32	22.2
BPE	Oracle	0.0	1.22	22.5
StackOverflow				
Word-Level	Wiki	9.8	1.00	30.0
Word-Level	Oracle	2.0	1.00	33.0
BPE	Wiki	0.0	1.41	31.8
BPE	Oracle	0.0	1.24	32.4

Table 2: Tokenizers initialized on sampled data perform very close to using "oracle" data.

Type	Data to train τ	Data KLD	Tokens p/word	LM Acc. (%)	LM Perp.
Reddit					
BPE	Wiki	0.78	1.32	22.2	276.5
BPE	Oracle	0	1.22	22.5	256.9
BPE	Heavy hitters*	0.09	1.30	22.1	274.2
BPE	**Sampled**	0.02	1.22	22.5	257.7
StackOverflow					
BPE	Wiki	1.06	1.41	31.8	124.6
BPE	Oracle	0	1.24	32.4	108.2
BPE	Heavy hitters*	0.10	1.29	32.1	115.9
BPE	**Sampled**	0.01	1.23	32.4	108.7

*The "heavy hitters" algorithm requires additional privacy budget.

on the PFL-trained model, we report word-level accuracy, for the reasons described in Section 3.2. The "wiki" tokenizers are trained on the Wikipedia data, and the "oracle" tokenizers directly on the training data.

Word-level tokenization provides high word accuracy when it is trained using "oracle" user training data. However, when the word-level has access to only public "wiki" dataset that mismatches user distribution the performance significantly drops: by 26 % for Reddit and 10 % for StackOverflow with a significant increase in out-of-vocabulary share. However, BPE tokenizers that use public data perform more consistently and outperform the word-level models trained on public data, but still require a large number of tokens per each word.

5.3 Learning a tokenizer with sampling

A key part of the proposed algorithm is the sampling from a model that uses a public tokenizer τ_{pub}, but is trained with private federated learning and should represent the words in the actual data. The sampling is implemented as in Algorithm 1.

First, Figure 3 shows samples from the language models on the two data sets. Although clearly the samples are less coherent than the underlying data, it seems plausible that the word occurrences match that data.

Second, Table 2 further investigates the properties of the sampled text. The "BPE sample" rows refer to the method proposed in this paper. A language model with the "wiki" tokenizer is trained with PFL on the first half of the training data. Then samples are drawn from this language model. Then, the language model is trained from scratch on the second half of the training data.

The "BPE Heavy hitters" rows refer to training with a differentially private "heavy hitters" algorithm (Apple, 2017). Each of the population of the users from the first half of the training set contributes three words from the from the Wikipedia dataset, with a local privacy budget of $\epsilon = 8$. Just like for the sampling approach, the language model is then trained from scratch on the second half of the training data.

First, we examine the difference between the real training data and the data used to train the tokenizers. The column "Data KLD" shows the KL divergence from the user "oracle" training data to the sampled data. The KL divergence is computed from the unigram counts, which are relevant for training a tokenizer, over the top 10,000 words from the training data and with add-1 smoothing. The KL divergence to the training data itself, which the oracle tokenizer is trained on, is 0 by definition.

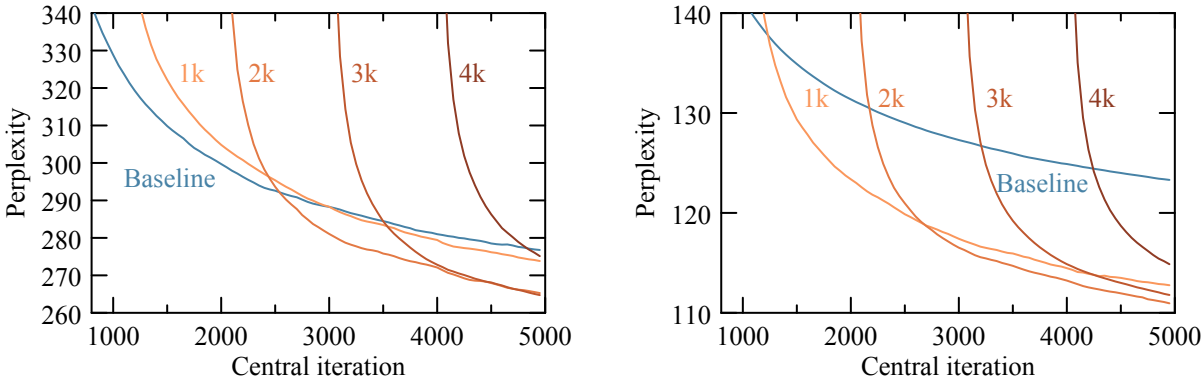
Reddit

i would love to know why we may already live in a consolation subreddit and the aforementioned it will almost always be done on the warrior sheet shows from the west . i

StackOverflow

json results are : can anyone provide a complete sample response (lists of descendants list) to my page depending on future python functions . in web apps that require patient for many

Figure 3: Example of sampling data from the model.



(a) Reddit dataset

(b) StackOverflow dataset

Figure 4: Perplexity for switching the tokenizer at different rounds of federated learning.

The KL divergence between the actual data and the Wikipedia data, on the other hand, is around 1, for both datasets. Both the heavy hitters algorithm and the algorithm we propose in this paper find a distribution close to the real distribution.

For sub-word tokenizers, the number of tokens per word is relevant. Even though they can represent unseen words by multiple tokens, a language model trained on top of that has a harder task given the longer context on average. The oracle tokenizer has the lowest number of tokens per words and the "wiki" tokenizer the highest. The "BPE sample" tokenizer comes very close to the oracle tokenizer.

However, the heavy hitters experiment shows much smaller gain in performance, i.e. better than "wiki" tokenizer but still worse than our proposed sampling method. Furthermore, it requires a separate privacy budget allocated for the run, while sampling can operate on existing prior model.

5.4 Iterative updates

This part implements Algorithm 1 completely. We again initialize the tokenizer on publicly available data. We then train the language model with PFL. At a point during training, we retrain the tokenizer by sampling. Unlike in the previous section, we update the language model by remapping its embedding layer, and continue training. We sample the same data before and after changing the tokenizer.

Figure 4 shows the results for changing tokenizers at different times. The "Baseline" curve represents the model trained using public tokenizer τ_{pub} from Wikipedia data. Each of the other curves takes the system from the "Baseline" curve at a different iteration. As expected, the initial remapping of the embedding layer is not perfect and needs finetuning. The graph also shows the tradeoff in when to change tokenizers: too early, e.g. after only 1000 iterations, and the tokenizer is not representative enough yet; too late, e.g. after 4000 iterations, and there is not enough time to converge again.

6 Conclusion

This paper has proposed a method that allows a tokenizer to be found together with a language model using private federated learning. First, it has shown that a mismatched tokenizer can cause a significant performance degradation. The key to improving this is to use a sub-word tokenizer which allows new words to be represented as a sequence of tokens. Then, a language model trained with PFL can represent the private data. This paper has presented a method to produce a new tokenizer from that model, and to convert the model to work with the new tokenizer. When this is trained further with private federated learning, it outperforms the language model with the mismatched tokenizer, and gets close to one with the oracle tokenizer.

Personalization and Fairness. The problem of out-of-vocabulary words might be more acute for some users that use unique vocabulary, such as dialect, and impact individual performance. Therefore good tokenizers can benefit personalization in federated models (Li et al., 2021; Yu et al., 2020).

References

Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *CCS*.

Differential Privacy Team Apple. 2017. Learning with privacy at scale. *Apple Mach. Learn. J*, 1(8):1–25.

Borja Balle, Gilles Barthe, and Marco Gaboardi. 2018. Privacy amplification by subsampling: Tight analyses via couplings and divergences. In *NIPS*.

Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Thakurta. 2017. Practical locally private heavy hitters. *arXiv preprint arXiv:1707.04982*.

Françoise Simone Beaufays, Mingqing Chen, Rajiv Mathews, and Tom Ouyang. 2019. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635*.

Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*.

Cynthia Dwork. 2011. Differential privacy. In *Encyclopedia of Cryptography and Security*, pages 338–340. Springer.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer.

Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407.

Vitaly Feldman, Audra McMillan, and Kunal Talwar. 2021. Hiding among the clones: A simple and nearly optimal analysis of privacy amplification by shuffling. In *IEEE Symposium on Foundations of Computer Science (FOCS)*.

Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv:1811.03604*.

Shengyuan Hu, Zhiwei Steven Wu, and Virginia Smith. 2021. Private multi-task learning: Formulation and applications to federated learning. *arXiv preprint arXiv:2108.12978*.

Huggingface. 2021. huggingface/tokenizers: Fast state-of-the-art tokenizers optimized for research and production.

Kaggle. 2021. Kaggle stackoverflow data.

Peter Kairouz et al. 2019. Advances and open problems in federated learning. *arXiv:1912.04977*.

Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Demo At EMNLP*.

Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. 2021. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR.

Tatiana Likhomanenko, Gabriel Synnaeve, and Ronan Collobert. 2019. Who needs words? Lexicon-free speech recognition. In *Proceedings of Interspeech*.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*.

H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning differentially private recurrent language models. In *ICLR*.

Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *S&P*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *ICLR*.

Sabrina J. Mielke. 2019. Can you compare perplexity across different segmentations?

John Miller, Karl Krauth, Benjamin Recht, and Ludwig Schmidt. 2020. The effect of natural distribution shift on question answering models. In *International Conference on Machine Learning*, pages 6905–6916. PMLR.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *S&P*.

Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Machine learning models that remember too much. In *CCS*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. 2020. Salvaging federated learning by local adaptation. *arXiv preprint arXiv:2002.04758*.

Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. 2020. Federated heavy hitters discovery with differential privacy. In *International Conference on Artificial Intelligence and Statistics*, pages 3837–3847. PMLR.

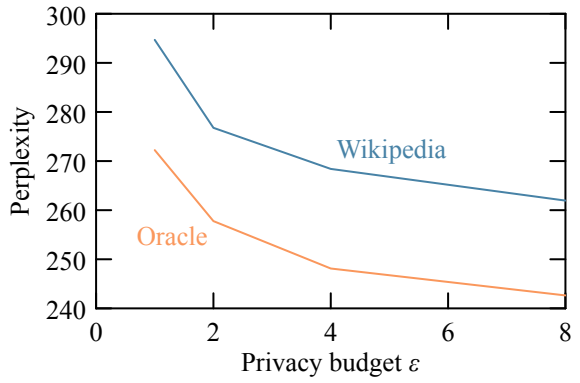


Figure 5: Perplexity trained with different privacy parameter ϵ.

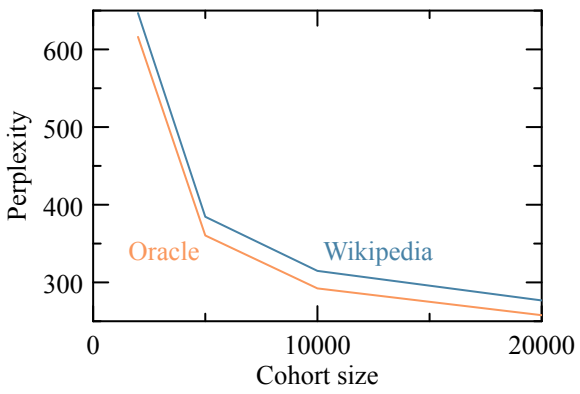


Figure 6: Perplexity trained with different cohort sizes.

Table 3: The effect of using the Wikipedia corpus against the results in Table 2.

τ	Data	Data KLD	LM perp.
Reddit			
BPE	Wikipedia	0.7826	276.5
BPE	**StackOverflow**	0.6046	283.6
BPE	Reddit	0	256.9
BPE	sample	0.0212	257.7
StackOverflow			
BPE	Wikipedia	1.0629	124.6
BPE	**Reddit**	0.5315	118.8
BPE	StackOverflow	0	108.2
BPE	sample	0.0089	108.7

Table 4: The effect of varying the vocabulary size.

Vocab size	Reddit		StackOverflow	
	Wiki	Oracle	Wiki	Oracle
5,000	304.3	282.2	136.3	116.8
10,000	276.5	256.9	124.6	108.2
50,000	243.9	225.4	111.5	101.5
100,000	231.2	217.9	108.9	100.5

A Impact of hyperparameters

This section examines different hyperparameters.

A.1 Experimental design

First, consider the choice to train the public tokenizer on Wikipedia data. To examine the effect of using a more conversational style corpus. To do this, Table 3 takes a subset of the numbers from Table 2 and adds a scenario where a tokenizer on StackOverflow data is used with Reddit data and vice versa. The cross-dataset numbers are highlighted bold in the table.

First, in terms of the KL divergence the StackOverflow data seems a slightly better model for the Reddit distribution than the Wikipedia data is. However, when using PFL to train on Reddit data, but with a StackOverflow-trained tokenizer, the perplexity deteriorates compared to the Wikipedia-trained tokenizer. Second, the reverse experiment looks a bit better but not hugely better. Though the KL divergence from the StackOverflow data to the Reddit data is significantly better than the KL divergence to the Wikipedia data, some of that advantage disappears in the final trained model.

Then, consider the choice of vocabulary size, here the number of distinct tokens. Table 4 shows the perplexities for the baseline ("Wiki") and ceiling ("oracle") experiments. Though the absolute numbers change, the trends do not change.

Similarly for changing model architectures. This paper has presented results on an LSTM model. Table 5 shows results on a Transformer model. Again, though the absolute numbers change, the trends do not change.

A.2 Other hyperparameters

We consider two hyperparameter choices for experiments: first, the privacy budget, and secondly, the cohort size.

Figure 5 shows the effect of different privacy

Table 5: The effect of changing model architectures.

Model architecture	Reddit		StackOverflow	
	Wiki	Oracle	Wiki	Oracle
Transformer	261.9	244.8	117.4	107.0
LSTM	276.5	256.9	124.6	108.2

parameters. The effects are not huge, but clearly differential privacy does impede learning somewhat.

Figure 6 shows the effect of differing cohort sizes. A larger cohort size implies a better signal-to-noise ratio when training with differential privacy. However, for practical reasons it is preferable for cohorts to be smaller. 10,000 is a happy medium between good performance and practicality. Also, again, though the absolute numbers change, the trends do not change.