Exploring a Bilingual Next Word Predictor for a Federated Learning Mobile Application

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

08-01-2022 / 23-02-2022

CITATION

Burgos, Natali Alfonso; Kiš, Karol; Bakarac, Peter; Kvasnica, Michal; Licitra, Giovanni (2022): Exploring a Bilingual Next Word Predictor for a Federated Learning Mobile Application. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.18058682.v2>

DOI

[10.36227/techrxiv.18058682.v2](https://doi.org/10.36227/techrxiv.18058682.v2)

Exploring a Bilingual Next Word Predictor for a Federated Learning Mobile Application

N. Alfonso Burgos*, K. Kiš†, P. Bakarac‡, M. Kvasnica§, G. Licitra¶

*¶Neurocast B.V., Amsterdam, the Netherlands

†‡§Slovak University of Technology in Bratislava, Bratislava, Slovakia

Email: *natali@neurocast.nl, †karol.kis@stuba.sk, ‡peter.bakarac@stuba.sk, §michal.kvasnica@stuba.sk, ¶giovanni@neurocast.nl

Abstract—We explore a bilingual next-word predictor (NWP) under federated optimization for a mobile application. The predictor of choice is a character-based LSTM trained on English and Dutch texts from a custom parallel corpora. Here, we assess the feasibility of distributed training in a simulated federated learning environment, where the popular Federated Averaging (FedAvg) algorithm implements the aggregation step. We show that the federated LSTM achieves a decent performance, yet sub-optimal compared to its server-trained counterpart. We suggest further steps to taper off the performance gap. Furthermore, we explore the effects of language imbalance varying the ratio of English and Dutch training texts. We show the predictor upholds performance (of the balanced case) up and until an 80/20 imbalance before decaying rapidly. Lastly, we describe the implementation of local client training, word prediction and client-server communication in a custom virtual keyboard for Android platforms. Homomorphic encryption secures the aggregation step guarding the user against malicious servers.

Index Terms—Federated Learning, Bilingual Word Prediction, Character-based LSTM, Homomorphic Encryption

I. INTRODUCTION

In 2019, the Global Mobile Consumer Trends survey reported smartphones to be the most ubiquitous electronic device in developed markets: about 90% of consumers own one and use them daily [1]. What is more, instant messaging applications are consumers' favourite, thus stirring a fair share of efforts into creating smoother and faster texting experiences, i.e. auto-complete and auto-correction features, word and emoji suggestions, and in-app speech-to-text. Now, for the first time, access to continuous, uninterrupted and vast amounts of data is attainable, allowing AI-driven, data-hungry algorithms to power these applications that, in turn, promote more usage and more data to be created.

Consumer data is, however, private and often protected. Data privacy protection laws in the European Union - General Data Protection Regulation (GDPR) - restrict the application of AI algorithms on personal data and dictate that its collection “[...] shall be adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed” (data minimisation principle) [2]. Federated Learning (FL) eliminates the need for data collection altogether and helps develop GDPR-compliant AI systems. More specifically, FL refers to the approach of learning a task by a

federation of user devices (*clients*) orchestrated by a central server [3]. This technique allows users to benefit from a shared global model without sharing data or centrally storing it. Oftentimes, data privacy methods like differential privacy [4], [5] or homomorphic encryption [6] are embedded in the system to improve security further. FL provides a promising alternative to server-based data collection and model training in commercial settings.

It is important to note that costs can escalate quickly in such a system. One must carefully consider CPU usage, memory footprint, battery consumption and network bandwidth, to name a few, when locally training and updating models in FL systems. Hence, on-device training of language models is limited in commercial mobile keyboards. Thus, language models are usually constraint to tens of megabytes in size to be able to run, even on high-end devices, while delivering predictions at a reasonable latency (within 20 milliseconds of an input event) [7]. These limitations come at the cost of prediction accuracy, the size of vocabulary, and multilingual capabilities.

In this paper, we investigate the possibility of a one-shot bilingual Next-Word Predictor (NWP) for a mobile application in a federated learning fashion. We simulate an FL environment where we train an RNN-based model from scratch in two languages with a shared alphabet (English and Dutch). In addition, we investigate the effects of language-specific sample imbalance in learning more than a single language.

Additionally, we provide the implementation details of the custom mobile application of our federated NWP. Here, model updates are performed on encrypted parameters using homomorphic encryption.

A. Related Work

Besides data privacy benefits derived from FL by design, others also emerge regarding resource consumption. Offloading computationally-intensive operations like the training of Deep Learning (DL) models to edge devices is an incredible perk. However, due to hardware and data transfer limitations, models must be kept small and concise. State-of-the-art Language Models (LMs) are primarily self-attention based, so-called *transformers*, namely of tens of millions of parameters,

and many count with multilingual capabilities (e.g., BERT is trained on 104 languages) [8]. Even after optimizing for size and inference latency with methods like network pruning, weight sharing, Knowledge Distillation, and quantization, models remain on the heavy side [9]. Hence, they are suitable for on-device, offline inference but not FL settings.

Applications of FL on Natural Language Processing (NLP) tasks have gained considerable attention in recent years [10]. Access to real-world text data, and the decentralization of computation and storage, all while preserving privacy, acted as great incentives to this motive. Emoji prediction [11], query suggestions [12] and next-word prediction [7], [13]–[15] are instances of NLP tasks solved on FL environments (on-device and simulated), to name a few.

Here, we focus on next-word prediction in an FL setting. Like in [13], we train a character-level RNN-based language model, except with bilingual capabilities.

B. Federated Learning

FL is an approach to distributed machine learning that builds privacy into infrastructure by design. It consists of a series of *training rounds*, where a central server coordinates the number of *clients* that participate in a round. Each client starts off with the same global model and computes a single or several local updates to it, e.g. an epoch of mini-batch stochastic gradient descent. Model updates are performed using local, client-generated training data and are never sent to or collected centrally in the server. After training is completed, updates are communicated to the server. The server aggregates the contributions and incorporates them into the global model. Note that updates are ephemeral and only live until immediately after aggregation. The new global model is sent back to the clients, along with the minimal information necessary for model training [3], [10], [12]. A graphical representation of FL is shown in Figure 1.

Aggregation methods have been actively researched in recent years (an exhaustive overview of existing methods can be found in Tables 1 and 2 in [10]). FedAvg algorithm is currently the most widespread method in FL settings, reported in over 58 publications by the end of 2020 [10]. FedAvg updates the server-side model using the parameters (or weights) of the model through a weighted average of participating client models defined as follows

$$w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k \quad (1)$$

where w_{t+1}^k are the model parameters of the k^{th} client.

In comparison to its predecessor, FedAvg has shown to converge faster, reducing the number of training rounds by 90% and working surprisingly well when client models share the same initialization [3]. Because of its effectiveness and simplicity, we consider FedAvg as the FL algorithm.

Lastly, and more generally, FL is a distributed approach to an optimization problem. The implicit problem to solve in an FL setting was coined by [3] as *federated optimization*. Alongside communication constraints and many other practical issues, they made emphasis on the unbalanced and non

Fig. 1: Federated Learning Diagram (i) Initialize global model and send them to client devices (ii) train model on client devices with client-generated data (iii) each client sends the trained model back to the server once training is completed (iv) once the number of client models for a training round is met, the server aggregates them (v) the global model is updated, and a new round of training begins.

Independent and Identically Distributed (IID) nature of data distributions in federated optimization. FedAvg applications showed to be robust under these conditions. In the present work, we are especially interested in non-IID distributions, not of target symbols, but across input languages. Statistical distributions of characters from English and Dutch texts are comparable, yet the effect on performance accuracy and speed of convergence of unequally distributed, language-specific texts across clients is yet to be disclosed.

C. Homomorphic Encryption

An alternative is to provide with a secure aggregation to the central averaging server that is capable of computing, at each iteration t , the average model w_t in Equation (1) without explicit knowledge (hence, of interpretation) of individual client model parameters w_t^k . This can be achieved by employing the concept of *homomorphic encryption* [16], [17]. In short, homomorphic encryption allows to perform mathematical operations over ciphers of plaintext data. Consider an encryption mechanism $\mathcal{E} : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ and a decryption function $\mathcal{D} : \mathbb{N}_0 \rightarrow \mathbb{N}_0$, whereby $c = \mathcal{E}(x)$ is the cipher of a non-negative integer x , and $\mathcal{D}(c) = x$ is the decryption operation. Naturally, $\mathcal{D}(\mathcal{E}(x)) = x$. It is important to note that functions $\mathcal{E}$ and $\mathcal{D}$ are asymmetric; in the sense that encryption is performed using a public key that allows any interested party to generate cipher $\mathcal{E}(x)$, yet the decryption key is kept private granting permission only to its legitimate owner to apply the decryption operation $\mathcal{D}(\mathcal{E}(x))$.

Remark 1: Although the encryption and the decryption functions are defined for non-negative integers, they can easily be extended to support floating-point numbers by first shifting them to the range $[0, \infty)$ and subsequently converting them to integers, e.g., by applying quantization. In what follows, we

Fig. 2: Language Character Distributions. Character frequency distributions of (a) English and (b) Dutch texts in the training set.

LSTMs use purpose-built, gated memory cells that store historical information of a sequence and ensure correct propagation of information through many time steps [23]–[25].

Albeit deterministic, LSTMs can be used to learn probability distributions over a sequence of language symbols. The joint probability over symbols is defined using the chain rule,

$$Pr(S = \{s_1, \dots, s_N\}) = \prod_{i=1}^N Pr(s_i | s_1 \dots, s_{i-1}) \quad (2)$$

where the current symbol is conditional on the previous one, and where the context of previous symbols is encoded in the hidden states of LSTMs. The conditional probability $Pr(s_i | s_{i-1})$ is, therefore, a multinomial distribution parametrized by a softmax function.

For LMs, the goal is to maximize the log-likelihood of a given sequence of symbols by optimizing the cross-entropy (CE) between the prediction and the target probabilities. However, computing the joint probability over a vast number of symbols becomes prohibitively slow during training [26], [27]. Character-level language modelling is an exciting alternative to word-level or subword-level next-word prediction. Not only are the set of symbols much smaller in size, but it allows for the inclusion of Out-Of-Vocabulary (OOV) words.

We employ a character-level LSTM with multiple layers to generate the next word to this extent.

II. METHODOLOGY

A. Dataset

The dataset is a collection of parallel corpora¹ (English-Dutch) accessible from OPUS open parallel corpus website².

¹A parallel corpus is a large and structured set of translated texts between two languages.

²<https://opus.nlpl.eu/>

Corpora come in Moses format³ and proceed from various sources:

- **OpenSubtitles** (37,2M) a collection of translated movie subtitles from *OpenSubtitles.org*⁴ [28].
- **TED2020** (319,9K) contains a crawl of nearly 4000 TED and TED-X talks transcripts from July 2020. A global community of volunteers translated the transcripts to more than 100 languages [29].
- **Wikipedia** (797,1K) a corpus of parallel sentences extracted from *Wikipedia* by K. Wolk and K. Marasek [30].
- **QED** (411,3K) an open multilingual collection of subtitles for educational videos and lectures collaboratively transcribed and translated over the *AMARA* web-based platform [29].
- **Books** (38,6K) a collection of copyright free books aligned by A. Farkas⁵ [29].

Corpora are stripped of HTML tags, URLs, citations, titles, and subtitles, and the remaining text is split into sentences (one per line). The aftermath is 69,7M lines of text data. Character frequency distributions among both languages are comparable (Figure 2). There are 76.9 ± 69.3 characters per sentence in English and 92.5 ± 72.3 in Dutch. Sentence length variability is large and right-skewed.

B. Data Partition and Distribution

We hold out 5% of each parallel corpus as the test set. Namely, parallel sentences pertain to the same set to avoid possible lexical leakage. This process results in 69,7M training lines and 7.7M testing lines. Note that both sets contain the same number of Dutch and English sentences.

³Two aligned plain text files.

⁴<http://www.opensubtitles.org/>

⁵Available on http://www.farkastranslations.com/bilingual_books.php

Provided that the data is not user-tagged (at least not all of it), training samples are sharded at random and unequally into K client shards to simulate an unbalanced and non-IID client dataset. Each client shard is further partitioned into R equally-sized parts to mimic the local client cache consumed in every training round (Figure 3). Sharding is performed separately for each language.

We adjust the fraction F of language-specific clients in order to study the impact of *language imbalance* on federated optimization. More specifically, F is the fraction of English-speaking clients (EN clients) in the training data, and thus $1 - F$ is the fraction of Dutch-speaking clients (NL clients). This results in $K \times F$ EN clients and $K \times (1 - F)$ NL clients (Figure 8 in Appendix).

C. Data Processing

Given that Dutch and English languages employ the same (Latin) alphabet, the size of the symbol set $M = |S|$ (or the vocabulary size) is reduced to $M = 69$ unique characters: lowercase Latin characters (26), numbers (10), punctuation symbols (32), white-space and $\langle \text{UNK} \rangle$ special tag. Characters in sentences are standardized to lower-cased ASCII characters, and unknown characters are replaced with $\langle \text{UNK} \rangle$. Sentences are truncated at the length of 100 characters, as it has repeatedly shown the best results. Shorter sentences are pre-padded with zeros, as suggested in [31]. Lastly, we replace characters with their index representations. Indices are obtained by enumerating the set S , such that $f : S \rightarrow J$,

Fig. 3: Client Cache Sharding. The client dataset consist of 69.7M English and Dutch training sentences (yellow and green), found at equal parts, partitioned into K unequally-distributed client sets (yellow). Note that a set $\{k_i\}_{i=1:K}^\ell$ exists per language, ℓ . In turn, local cache is simulated by sharding each client $k_i \in \{k_i\}_{i=1:K}^\ell$ into R equally-sized parts (blue). Each client local cache $k_i^r \in \{k_i^r\}_{r=1:R}^\ell$ is consumed by client model i at round r . **Glossary:** number of languages (L), number of clients (K), number of training rounds (R).

where $J = \{j_1, \dots, j_M\}$ is the index set and f is the particular enumeration of S .

Sentences are fed to the model as sequences of *one-hot* character vectors. Namely, every character index, j , is represented by a zero-valued M -dimensional vector, whose element on index j equals one. A *language code* is concatenated to the character vector. This is encoded as a binary vector of length $\lceil \log_2(\ell) \rceil$, where ℓ is the number of languages. In a bilingual setting, $\lceil \log_2(\ell) \rceil = 1$. The resulting input sequence is of size $M + \lceil \log_2(\ell) \rceil = 69 + 1 = 70$.

Last but not least, we build targets by forward-shifting input sequences. Essentially, any given character acts as the target to the previous character in the sequence.

D. Model Architecture

The model is a multi-layer LSTM, with six (6) stacked LSTM layers of 128 hidden neurons each. It counts with over 771K parameters and is 3MB in size. If weights are quantized, the model size drops as low as 0.8MB. Models large in width (1024+ neurons per layer) with about half the depth performed better in some instances, however, with many times more the size. A deep but moderately wide model results in a more convenient trade.

E. Federated Optimization

The multi-layered LSTM is at the core of the federated next-word predictor. The *Federated LSTM*, or **F-LSTM**, trains a global LSTM model distributively over K clients, where only a fraction C of clients are considered on each training round by the *FedAvg* algorithm. The global model is trained for a maximum of R rounds. Clients update the global model after a training round is closed by averaging their model parameters (see Equation 1). Local training at client nodes consists of E epochs of $\frac{n_k}{B}$ mini-batch Stochastic Gradient Descent (SGD) optimizations with momentum, where n_k is the number of local samples generated by the k^{th} client, and B is the size of the mini-batch. Momentum is computed following Nesterov's accelerated gradient method [32]. Momentum-based optimizer have shown to work best on fully non-IID data-sets in federated learning settings [33]. Additionally, to avoid exploding gradients in BPTT, gradients are clipped when their L2-norm exceeds a maximum value of 2. Gradient clipping increases the robustness of model convergence.

In FedAvg, client and server learning rates are decoupled by formulating the server update as if applying SGD to the "pseudo-gradient", $-\Delta_t = \sum_i^K (w_t^i - w_t)$ [34]. This formulation is often used to increase the degree of freedom of the system for better convergence rates. However, as mentioned in [34], setting the client learning rate has a greater impact on converge. For that, and simplicity's sake, we opt to not decouple learning rates (i.e., $\eta_{server} = 1.0$) and tune a single (client) learning rate η . The client learning rate is decayed by

$$\eta = \frac{\eta_0}{1 + t_c} \quad (3)$$

where η_0 is the initial value of η , and t_c is the times η has been decayed. Counter t_c increases by a unit if the following condition is met

$$\ell_t \leq \min_{j < t} \ell_j - \Delta \quad (4)$$

where ℓ_t is the average training loss in round t , and $\min_{j < t} \ell_j$ is best average training loss (relative to Δ) across an arbitrary number of previous rounds [35], [36].

F. Prediction Sampling

Fig. 4: Top-k Prediction Sampling. The model samples from the top- k most probable symbols at every time step t .

During inference, the next character is picked based on the joint probability over characters defined in Equation 2. Rather than selecting the character with the highest probability, the system draws one from the top-3 most probable characters to add a higher degree of flexibility to the language model. This strategy is customary in language modelling (see Figure 4 as an exemplary).

The sampling process begins with a character that acts as a *starts token*. Start tokens can be a sequence of characters, too. At each step t , the model samples a character based on $Pr(s_t | s_{0:t-1})$. This is done iteratively until a *stop symbol* is encountered (e.g. white-space, $\langle UNK \rangle$, $\langle PAD \rangle$, $\langle EOS \rangle$) or the sequence reaches a maximum of number of characters.

During testing, sampling is carried out for every sentence independently. Namely, we initialize hidden cell states at the beginning of every sentence prediction.

G. Performance Evaluation

The quality of the federated model is tested against the hold-out set described in Section II-B. We test after every training round is closed. Client models are not tested against the hold-out set: simulations are run serially on a single-GPU machine, and thus testing client models become highly time-consuming. Nonetheless, we report their training performance.

At *character-level prediction*, we are interested in assessing the learning process concerning a target performance.

Therefore, we consider *Central LSTM*, or **C-LSTM**, the traditional centralized counterpart of the same model definition as our benchmark. C-LSTM allows optimization strategies like training for multiple epochs, online validation, early stopping, adaptive learning rate methods, and learning rate scheduling that increase the chances for optimal convergence. C-LSTM yields a Cross Entropy (CE) loss of 1.37, and an overall/top-3 accuracy of 77.61 % and 88.42 %, respectively. We report accuracy, precision, recall and bytes-per-character (BPC).

III. EXPERIMENTS

A. Setup

The experimentation follows a two-stage process: (A) the optimization of the system, which comprises the hyper-parameter tuning of the federated system to achieve the best performance and speed of convergence possible, (B) the experimentation of various ratios of EN/NL clients, or values of F , and their effect on performance and convergence. In this series of simulations, F takes on the following ratios $F = \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. Note that the default language imbalance of the training data is $F = 0.5$. In any case, client training and global model evaluation are executed concurrently and independently.

Some parameter settings are shared across all experiments to adapt to hardware and time constraints. We simulate $K = 600$ clients for a maximum of $R = 1000$ rounds with a client participation ratio of $C = 0.1$. Namely, 10% of clients, selected at random, are allowed to participate in every training round⁶. We decay the learning rate η following Equation 3 if no progress is made (relative to $\Delta = 0.001$ in Equation 4) after 5 consecutive training rounds. We implement early stopping under the same criterion, except that training is interrupted after 15 consecutive rounds. The same seed is used for all experiments.

B. Environment

We employ an AWS EC2 instance (p2.xlarge) equipped with a single NVIDIA Tesla K80 GPU with 12GiB of memory, four (4) vCPUs, 61GiB of RAM and EBS-backed storage of 100GB. Client simulations run serially provided that we are limited to a single GPU instance.

C. Results

Table I shows various goodness-of-fit measurements of F-LSTM and C-LSTM for English and Dutch languages, separately. A performance gap stands out between centralized and federated optimization of our NWP model. C-LSTM outperforms its federated counterpart by 13.98% in overall accuracy. Complementary to Table I, train and test CE loss progression over training rounds is illustrated in Figure 5. It shows that, on average, F-LSTM converges to a test CE loss of 2.82 and top-3 accuracy of 76.2 %. Convergence is achieved at round $R_t = 42$ with learning rate $\eta_t = 0.15$. Hyper-parameter settings are detailed in Table II. These settings return the best

⁶Note that during this sampling process F ratio is kept unaltered.

TABLE I: Table of Results. Performance metrics of best performing F-LSTM and C-LSTM with $F = 0.5$. Results are reported for English (EN) and Dutch (NL), separately.

Performance Metric	C-LSTM		F-LSTM	
	EN	NL	EN	NL
CE Loss	1.44	1.50	2.69	2.61
Accuracy	87.21 %	68.12 %	66.02 %	67.50 %
Top-3 Accuracy	93.54 %	81.75 %	70.52 %	72.82 %
Precision	87.21 %	75.54 %	66.02 %	67.50 %
Top-3 Precision	31.18 %	27.96 %	23.50 %	24.27 %
Recall	87.21 %	68.12 %	66.02 %	67.50 %
Top-3 Recall	93.54 %	81.75 %	70.52 %	72.82 %
BPC	2.08	2.16	17.57	17.05

model performance among many experimental iterations. In every case, exponential learning rate decay and momentum lead to a faster and more stable convergence rate, respectively. Moreover, Table I shows high rates of top-3 accuracy and recall (70%-95%) in both optimization cases, while precision rates remain noticeably lower (20%-35%). In a multi-class classification problem, high recall and low precision suggests that there is a preference for the dominant classes (*white-space*, *e*, *a*, ...; see Figure 2), and thus the model is under-fitting the hold-out set. However, when considering the overall distribution, the model is more precise yet at the cost of accuracy.

In terms of model diagnostics, Figure 5 discloses the relation between train and test CE loss. They go hand-in-hand over training rounds suggesting that F-LSTM either (1) needs further training to reach the target performance or (2) that it has converged to sub-optimal local minima of the global cost function.

Figure 4 in [13] showcases the CE loss of a character-level NWP over 3000 training rounds during a live client evaluation of various FL settings. Readily, in the first rounds, it becomes evident that FedAvg plateaus showing hardly any further improvements, yet very high recall and precision rates are reported. It is well-known that CE loss is sensitive to label imbalance (e.g. [37]). Language character datasets for NWP are naturally and purposely imbalanced as depicted in Figure 2. CE loss penalizes unlikely events, even if these events are legitimately so. Hence, under a similar large CE loss, many more training rounds could lead to qualitatively better models.

Alternatively, and most likely, F-LSTM has converged to an inferior solution. Theoretical understanding of the (non-)convergence of real-world applications of FedAvg is to-date work in progress. Despite many efforts to tackle or minimize sub-optimality in FedAvg, [35], [36], [38]–[40] to name a few, there is no theoretical guarantee that it consistently improves upon mini-batch SGD local updates for non-convex objectives and heterogeneous (non-IID) settings. Sub-optimality is exacerbated by practicalities of FL networks, for instance, the number and selection strategy of effective devices contributing to training [35], [36].

Empirically, FedAvg has shown to be a promising estima-

tor of the global objective [3]. Nonetheless, the performance gap between centralized and distributed training with FedAvg is large for NLP tasks in particular. [38] pointed out the latter, where they compare the performance of various distributed NLP tasks with different FL algorithms and their centralized training. They report large performance gaps in accuracy across FL algorithms (Table 2 in [38]). They also noted that FedOpt [41], where client (ClientOpt) and server (ServerOpt) optimizers are decoupled, and ClientOpt is AdamW, shows the best accuracy across the board. Their report is in congruence with our centralized training: adaptive optimizers worked best in comparison to vanilla and momentum SGD. Lastly, they studied the impact of different degrees of data heterogeneity in terms of label (statistical heterogeneity, non-IIDness) and quantity (sample imbalance) distribution. They showed that statistical heterogeneity has important consequences to the convergence of FedAvg. However, for a uniform distribution of labels across clients, the convergence remains sub-optimal for some NLP tasks (particularly, for Language Modeling tasks).

On the one hand, it is generally preferred to offload computation to client nodes, i.e. adding more local SGD updates per round and reducing the number of communication rounds to reach a target performance. Offloading is achieved by either decreasing B , increasing E or C , or any combination of the three. Yet, allowing for more local SGD steps per round directs the convergence of FedAvg to an inferior optimum in non-IID settings. The phenomenon is known as "*client drift*", and it refers to the notion that local objectives may differ drastically from the global objective. Thus, over-fitting local objectives can result in biased global model updates. Figure 7, in the Appendix section, illustrates the degree of statistical heterogeneity (Jensen-Shanon distance) among participating clients on the first round. Overall, clients show low statistical heterogeneity, except for a few highly heterogeneous client pairs (in blue). We believe that these clients may be driving the drift. Mitigation strategies such as regularization [39], [40], learning rate decay schemas [36] or control variates [42], [43] can help minimize client drift, and in some cases, harnesses FedAvg to optimal convergence. All the same, in the stochastic non-convex case, convergence is not guaranteed.

We set $E = 1$, $B = 16$, $C = 0.1$ and $R = 1000$ in an attempt to find an optimal trade-off between client drift and computation efficiency, as suggested in [3]. These settings returned $K \times C = 60$ clients per round, and 3.39 ± 2.2 local steps in average (61.84 ± 35.11 average samples per round). Larger batch sizes $B > 16$ causes FedAvg to plateau or diverge. This is in line with the intuition that smaller local step sizes have a regularizing effect while increasing gradient variance [36], [44], and in turn, mitigating client drift. Increasing $E > 1$ instead accelerates convergence to the same sub-optimal point at the risk of over-fitting. As pointed out in [35], as long as we decay the learning rate, FedAvg can converge to the optimal even if $E > 1$. The same holds for C . FedAvg converges to the same point whatever the value of C , which allowed for low participation and quicker training rounds. However, for large enough C , the convergence is faster (similarly to [35]).

Fig. 5: Train & Test CE Loss over Training Rounds. CE loss of best performing F-LSTM for English (EN) and Dutch (NL) languages, separately. Top: Train CE loss across clients per round. Dashed lines report the average train CE loss per round. Bottom: Test CE loss per round. Overall test CE loss is outlined in solid red.

Differently, FedAvg is known to be robust against sample size imbalance. Recall that FedAvg computes the weighted sum of client models, where the weights are the normalized number of training samples, $\frac{n_k}{n}$. It allows clients to contribute to the global model proportionally to their local training data. In [38], they show that skewed sample size distribution over clients (drawn from a Dirichlet distribution for a range of β concentration factors) do not suppose a significant challenge to FedAvg. This was the case for F-LSTM, too.

Last but not least, we repeatedly increase the ratio of training English clients by 10%, and observe its effect on the convergence and performance of F-LSTM. Model accuracy for every case is collected in Table III, and test CE losses over training rounds are depicted in Figure 6. The latter shows a decline in model performance (increase in overall CE loss) proportionally to F (left). To further assess this behaviour, we tested F-LSTM against each language set, independently (bottom right: NL CE loss; top right: EN CE loss). As expected, NL CE loss monotonically increases with F , driving the overall performance decrease, while EN CE loss behaves in the opposite direction. Nonetheless, this effect is more severe in NL CE loss. The latter suggests that there might exist a minimum effective number of training samples or clients from where the performance stops decaying rapidly and starts improving slowly. Findings in Table III somewhat support this idea. Accuracy rates seem rather unaltered until a 20 % increase in F , when the performance drops drastically. Unlike CE loss comparatives in Figure 6, changes in accuracy are not gradual and complementary, but instead are abrupt and mirrored. An explanation is that the model is trained with the overall train CE loss, and so a decline or gain in

TABLE II: Hyper-Parameter Setting of the Best Performing F-LSTM for $F = 0.5$.

Hyper-parameter	Value
K^a	600
C^b	0.1
R^c	1000
R_t^d	42
E^e	1
B^f	16
η_0^g	1.5
η_t^h	0.15
m^i	0.9

^a number of clients

^b fraction of participating clients

^c number of maximum rounds

^d convergence round

^e number of epochs

^f batch size

^g initial learning rate

^h convergence learning rate

ⁱ momentum hyper-parameter

performance in any language sub-task unavoidably affects the other. The sensitivity of performance accuracy to small changes in CE loss reflects the non-linear nature of their relationship for unbalanced datasets. An observation on CE loss for performance assessment in Language Modelling tasks was mentioned earlier in this section.

IV. CLIENT-SIDE SYSTEM IMPLEMENTATION

This section provides a brief description of the proposed client-side system implementation. Inference, training and client-server communication are built within a custom soft keyboard in Android Studio.

The main feature of a soft keyboard is to test the entire framework word predictions as the user types. On-device, the prediction is supported by PyTorch Mobile [45] and, as the user sends or confirms the text input, the typed sequences are saved into a local cache. Predictions and text collection are disabled when the user types URIs, e-mail addresses, and passwords to protect the user's personal information. The reported time between the key-press and a word prediction is 11 ± 4 ms.

Client devices must meet several requirements to participate in the training process. Locally cached data must contain a minimal number of characters. The device must be charging, idle and connected to the unmetered network. After fulfilling these criteria, the device is ready to train the local model. Using Android's `JobScheduler` workers, we train the model and communicate with the server in the background. Training and encryption of the model are handled by CHAQUOPY [46]. CHAQUOPY is a Python SDK for Android that enables the use

Fig. 6: Test CE loss of F-LSTM with different language ratios. Test CE loss of F-LSTM for $F = \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. F-LSTM is tested against the overall (left), only English (upper right) and only Dutch (bottom right) hold-out sets.

of python training scripts to build PyTorch pipelines within an Android device.

A first worker executes the local training and performs a single epoch of mini-batch stochastic gradient descent. Model parameters are encrypted and stored in the device. After local training is completed, a second worker sends the encrypted model parameters to the server via a POST request. Immediately after, a third worker sends a GET request to obtain the global model and, once received, it decrypts it.

V. CONCLUDING REMARKS

This paper explores a bilingual next-word predictor mobile application using federated learning. Firstly, we centrally trained a single mobile-size character-level model to predict the next word either in Dutch or English. We show that we can successfully generate language-specific words given a language ID by continuously sampling the model. Next, we simulate a federated learning environment to assess the feasibility of distributed training of the same model. Our results show that the federated model converges to a sub-optimal point, affecting the model predictions' quality. Longer training rounds and/or decoupling client and server optimizers and learning rates are promising options to bridge the performance gap. Furthermore, varying the ratios of training English and Dutch clients disclosed how acute language imbalance deteriorates the performance of both sub-tasks.

Lastly, we describe the client-side implementation of the federated NWP using Pytorch Mobile, CHAQUOPY and Android Studio in a custom soft keyboard. Finally, we apply homomorphic encryption to securely perform the central averaging of clients models against malicious or dishonest central averaging servers.

REFERENCES

- [1] C. Deloitte, "Global mobile consumer trends," 2017.
- [2] "General data protection regulation (gdpr)," <https://gdpr-info.eu/>, (Accessed on 18/02/2021).
- [3] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," *ArXiv e-prints*, pp. arXiv-1602, 2016.

TABLE III: Table of Results. Performance metrics of best performing F-LSTM and C-LSTM with $F = 0.5$. Results are reported for English (EN) and Dutch (NL), separately.

Ratio of EN clients	Accuracy	
	F	EN NL
0.5 (50 %)		66.02 % 67.05 %
0.6 (60 %)		65.95 % 66.94 %
0.7 (70 %)		67.77 % 66.94 %
0.8 (80 %)		66.01 % 67.50 %
0.9 (90 %)		5.39 % 4.90 %
1.0 (100 %)		5.39 % 5.38 %

- [4] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Differentially private asynchronous federated learning for mobile edge computing in urban informatics," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 2134–2143, 2019.
- [5] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.
- [6] J. Zhang, B. Chen, S. Yu, and H. Deng, "Pefl: A privacy-enhanced federated learning scheme for big data analytics," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.
- [7] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.
- [8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- [9] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," 2020.
- [10] Y. Liu, L. Zhang, N. Ge, and G. Li, "A systematic literature review on federated learning: From a model quality perspective," *arXiv preprint arXiv:2012.01973*, 2020.
- [11] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," *arXiv preprint arXiv:1906.04329*, 2019.
- [12] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," *arXiv preprint arXiv:1812.02903*, 2018.
- [13] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays, "Federated learning of out-of-vocabulary words," *arXiv preprint arXiv:1903.10635*, 2019.

- [14] M. Chen, A. T. Suresh, R. Mathews, A. Wong, C. Allauzen, F. Beaufays, and M. Riley, "Federated learning of n-gram language models," *arXiv preprint arXiv:1910.03432*, 2019.
- [15] J. Stremmel and A. Singh, "Pretraining federated text models for next word prediction," *arXiv e-prints*, pp. arXiv–2005, 2020.
- [16] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *EURASIP Journal on Information Security*, vol. 2007, pp. 1–10, 2007.
- [17] X. Yi, R. Paulet, and E. Bertino, "Homomorphic encryption," in *Homomorphic Encryption and Applications*. Springer, 2014, pp. 27–46.
- [18] K. Laine and R. Player, "Simple encrypted arithmetic library-seal (v2.0)," *Technical report, Technical report*, 2016.
- [19] H. Williams, "A modification of the rsa public-key encryption procedure (corresp.)," *IEEE Transactions on Information Theory*, vol. 26, no. 6, pp. 726–729, 1980.
- [20] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.
- [21] J. D. C. Benaloh, "Verifiable secret-ballot elections," 1989.
- [22] W. De Mulder, S. Bethard, and M.-F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech & Language*, vol. 30, no. 1, pp. 61–98, 2015.
- [23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [24] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.
- [25] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent," *Backpropagation: Theory, architectures, and applications*, vol. 433, p. 17, 1995.
- [26] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.
- [27] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.
- [28] P. Lison and J. Tiedemann, "Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles," 2016.
- [29] J. Tiedemann, "Parallel data, tools and interfaces in opus," in *Lrec*, vol. 2012, 2012, pp. 2214–2218.
- [30] K. Wolk and K. Marasek, "Building subject-aligned comparable corpora and mining it for truly parallel sentence pairs," *Procedia Technology*, vol. 18, pp. 126–132, 2014.
- [31] M. Dwarampudi and N. Reddy, "Effects of padding on lstms and cnns," *arXiv preprint arXiv:1903.07288*, 2019.
- [32] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$," in *Dokl. akad. nauk Sssr*, vol. 269, 1983, pp. 543–547.
- [33] V. Felbab, P. Kiss, and T. Horváth, "Optimization in federated learning," in *ITAT*, 2019, pp. 58–65.
- [34] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," 2020.
- [35] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," 2020.
- [36] Z. Charles and J. Konečný, "On the outsized importance of learning rates in local update methods," 2020.
- [37] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *CoRR*, vol. abs/2001.00570, 2020. [Online]. Available: <http://arxiv.org/abs/2001.00570>
- [38] B. Y. Lin, C. He, Z. Zeng, H. Wang, Y. Huang, M. Soltanolkotabi, X. Ren, and S. Avestimehr, "Fednlp: A research platform for federated learning in natural language processing," *arXiv preprint arXiv:2104.08815*, 2021.
- [39] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *arXiv preprint arXiv:1812.06127*, 2018.
- [40] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *arXiv preprint arXiv:2007.07481*, 2020.
- [41] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *International Conference on Learning Representations*, 2021. [Online]. Available: <https://openreview.net/forum?id=LkFG3IB13U5>
- [42] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5132–5143.
- [43] S. P. Karimireddy, M. Jaggi, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Mime: Mimicking centralized stochastic algorithms in federated learning," *arXiv preprint arXiv:2008.03606*, 2020.
- [44] X. Qian and D. Klabjan, "The impact of the mini-batch size on the variance of gradients in stochastic gradient descent," 2020.
- [45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.
- [46] "Chaquopy python sdk for android," <https://chaquopy.com/chaquopy/>, accessed: 2021-06-10.

VI. APPENDIX

Fig. 7: Statistical Heterogeneity between Clients. Jensen-Shanon distance matrix between pairs of (a) 100 English clients and (b) 100 Dutch clients available in the first training round. Color intensity (towards blue) represents higher level of heterogeneity

Fig. 8: Language-Domain Partition. For $L = 2$, where $\{k\}^1$ and $\{k\}^2$ are the set of English and Dutch clients of size K , respectively, and $\{k\} = \{k\}^1 \cup \{k\}^2$ also of size K , we assign $F \times K$ English clients and $(1 - F) \times K$ Dutch clients to set $\{k\}$ for any given F . Here, we illustrate this partition strategy for $K = 600$ different values of F . Glossary: (K) number of clients, (F) fraction of English-speaking clients.