Training Mixed-Domain Translation Models via Federated Learning

Peyman Passban* **Tanya Roosta***
Rahul Gupta **Ankit Chadha** **Clement Chung**
Amazon
{peymp,troosta,gupra,ankitrc,chungcle}@amazon.com

Abstract

Training mixed-domain translation models is a complex task that demands tailored architectures and costly data preparation techniques. In this work, we leverage federated learning (FL) in order to tackle the problem. Our investigation demonstrates that with slight modifications in the training process, neural machine translation (NMT) engines can be easily adapted when an FL-based aggregation is applied to fuse different domains. Experimental results also show that engines built via FL are able to perform on par with state-of-the-art baselines that rely on centralized training techniques.

We evaluate our hypothesis in the presence of five datasets with different sizes, from different domains, to translate from German into English and discuss how FL and NMT can mutually benefit from each other. In addition to providing benchmarking results on the union of FL and NMT, we also propose a novel technique to dynamically control the communication bandwidth by selecting impactful parameters during FL updates. This is a significant achievement considering the large size of NMT engines that need to be exchanged between FL parties.

1 Introduction

Federated learning (FL) is a rapidly growing field in the machine learning community. The reason for this popularity is because of its *decentralized* and *private* nature. Model training in FL is distributed over multiple nodes where each node could be an independent piece of hardware with its own isolated data. This unique feature enables building high-quality models that benefit from external resources without requiring access to them.

Although FL is a relatively new field (McMahan et al., 2017), it has drawn researchers' attention and the community has witnessed a rapid growth. Fields such as computer vision have adapted quickly to the FL framework (Geyer et al., 2017; Lin et al., 2018; Hardy et al., 2019; Geiping et al., 2020; Ren et al., 2021), but others, such as natural language processing (NLP), have not been as quick and only recently have begun to explore (Wu et al., 2020).

We believe that the reason for this slow(er) integration of NLP and FL is because representation learning in NLP is complicated and downstream tasks require large and data-hungry models. These requirements are heavy-handed for any FL model and slow down the unification. However, real-world NLP problems necessitate the use of distributed solutions with privacy-preserving characteristics (Li et al., 2020). Our work is an effort towards leveraging FL-based solutions in NLP.

In this paper, we focus on NMT to combine it with FL. A review of the NLP literature shows that almost all recent groundbreaking architectures have been first proposed, or at least evaluated, for translation (Bahdanau et al., 2014; Sutskever et al., 2014; Gehring et al., 2017; Vaswani et al., 2017). This is an indication of the intricacy of the NMT task. Therefore, it is fair to claim that any FL technique that is capable of training high-quality NMT models could also be considered as a trustworthy alternative for other NLP tasks. Thus, NMT could be a strong candidate for FL benchmarking purposes.

NMT also has other unique features that can directly benefit FL. One key factor in a fair simulation of FL is to mimic data heterogeneity (also referred to as non-IIDness) in experimental environments (Kim et al., 2020). Different sampling techniques have been proposed to model such data distribution processes (Ji et al., 2020; Li et al., 2021; Wang et al., 2021), however there is no guarantee that what we simulate is what we encounter in real-world applications. NMT, to some extent, solves this problem since parallel datasets by nature have such heterogeneity. There are only a few NLP tasks that can provide as large and diverse training corpora as NMT. For some language pairs,

*Equal contribution.

there exist multiple datasets with hundreds of thousands or even millions of parallel sentences.[1] This means, we should have enough data for each FL node. Moreover, each node can naturally pick one dataset/language, so we do not have to artificially distribute data. NMT, together with offering rich training data, also has a bi-lingual setting which is a compelling testbed for FL. Coping with the complexity of not only different domains, but also different languages, at the same time in a distributed platform is worth investigating.

While NMT offers a natural testbed for FL, we argue that the task itself benefits from the offerings of FL. Training a mixed-domain translation model is a challenging task. We show that the aggregation phase of FL can greatly help with this challenge, as it efficiently fuses information from different domains. From the privacy perspective, NMT can also benefit from FL. In fact, the necessity of having a private training pipeline is relatively understudied in NMT. It is mostly assumed that all training datasets are available in a homogeneous and compatible format through a central repository, which is usually not attainable in real world. All these reasons make NMT a compelling case to study in the context of FL.

1.1 Research Scope

The goal of this paper is to provide preliminary results on the combination of FL and NMT, as opposed to running a comprehensive FL research or comparing different algorithms. We are also interested in studying the feasibility of training complex and deep NMT models in *decentralized* and *private* settings. Besides providing a set of benchmarking results, this paper's other two contributions can be summarized as follows:

- We show that FL aggregation techniques are reasonable alternatives to fuse information from multiple domains, thus FL-based training could be considered as an approach to build mixed-domain NMT engines.

- We show that large NMT models are hard to distribute within the FL network. Therefore, we propose a novel and cost-efficient solution to reduce the communication bandwidth.

[1] https://opus.nlpl.eu/

2 Federated Learning

FL is an approach to train models in a distributed fashion where nodes do not (and are not allowed to) access each other's data (Yang et al., 2019; Li et al., 2020). Any node by itself is not powerful enough to deliver high-quality services due to the small size of its local data. It can perform well on in-domain instances, but it might fail to respond to requests from other domains. FL establishes a communication methodology and a platform that allows participating nodes to exchange parameters (but not data) to help boost each other's performance.

In this work, we follow the *cross-silo* FL setting as outlined in Li et al. (2019). Algorithm 1 summarizes the entire model training pipeline and explains what we mean by being cross-silo.

Algorithm 1: Cross-Silo FL

```
for r ← 0 to rounds by 1 do
    updates = Pull(C)
    for upd in updates do
        S = aggregate(S, upd)
    C = Push(S)
```

In this setup, there is a central node $\mathcal{S}$ that orchestrates training. In each round r, the server pulls local updates (i.e. a set of parameters) from different nodes (also known as clients) and updates the parameters of the central model. $\mathcal{C} = \{c_1, ..., c_K\}$ indicates the set of K clients. Once all information is aggregated, parameters of the central model are pushed back to clients so that they can also benefit from global/community knowledge.

One key factor in FL is *communication*, which is defined by the `Pull` and `Push` steps in this algorithm. Due to the distributed nature of FL, nodes need to connect and exchange information and this needs to be carried out in an efficient fashion. The communication cost becomes even more critical when exchanging large models, such as in NMT. In the next sections, we discuss how communication directly affect the feasibility and performance of any FL setting, and how we improve it by our *dynamic pulling* technique.

Algorithm 1 only shows the computation that occurs on the server side. It should be noted that each client is an independent silo that updates its internal model with local data. This algorithm only illustrates the main skeleton of the cross-silo setting and does not entail all the details of each step.

In our experiments, we use the well-known FedAVG algorithm (McMahan et al., 2017) for *aggregation*. Therefor, Line 4 can be formulated as in Equation 1:

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

where w_r is the set of all parameters of the central model in the *r-th* round, n_k is the number of data points in the *k*-th client's dataset, and n is the total number of all training samples. There exist multiple extensions to FedAVG, but since it is a widely-acceptable baseline for benchmarking purposes we also use it in our experiments. This choice allows us to minimize the impact of different factors introduced by other FL algorithms and only focus on the relation between NMT and FL and their mutual impact on each other.

3 Federated Learning for NLP

There are several models in the field that have been proposed to leverage FL for NLP. Hartmann et al. (2019) studied whether they could improve the ranking of suggestions in the Firefox URL bar and train a model on user interactions without violating user privacy. They incorporated feedback received from different clients using FedAVG which resulted in significant improvements. Ji et al. (2019) suggested that the simple averaging strategy used in FedAVG might not be sufficient enough, so they improved the aggregation phase by incorporating the significance of each client by using an attention mask to weigh clients. Chen et al. (2019) focused on language modelling and addressed the problem of out-of-vocabulary entries when working with different clients.

Bui et al. (2019) investigated the effect of FL in training better and more personalized user/data representations. Their results show that when aggregating information via FL, the model quality increases significantly; at the same time, the training pipeline is distributed and private. We also make a similar observation in our experiments, though not at the representation level, but in terms of the final translation quality.

Ge et al. (2020) proposed a named-entity recognition (NER) model that is trained with FL to work on medical data. Their results demonstrate that not only FL preserves privacy, but also outperforms models trained in a centralized fashion. Apart from NER, models with similar concerns have been proposed for mobile keyword prediction (Hard et al., 2018), keyword spotting (Leroy et al., 2019), and next word prediction (Stremmel and Singh, 2020).

Liu and Miller (2020) utilized FL to pre-train and/or fine-tune BERT models (Devlin et al., 2018). From a research standpoint, it is worthwhile to understand if it is even feasible to handle such deep models in an FL framework, and whether a simple averaging-based aggregation is enough. They attempted to address these questions and provided supporting results. In that sense, our work is similar to theirs as we also work with complex models. In addition, we also discuss the bandwidth problem to facilitate exchanging large sets of parameters.

3.1 Domain Adaptation

Domain adaptation covers a wide range of problems from adjusting a model to work in a new domain/genre (Chu and Wang, 2018) to fine-tuning for noisier conditions (Passban et al., 2020), or even transferring a model to a different environment for a different task (Zhu et al., 2020). Domain adaptation has recently attracted more attention due to advances introduced by models such as ELMO (Peters et al., 2018) and BERT (Devlin et al., 2018). These models provide general-purpose representations which are easily adaptable to other tasks. In these models, *all* network parameters are fine-tuned during adaptation, which might not be necessary. Houlsby et al. (2019), Pfeiffer et al. (2020), and Rücklé et al. (2020) proposed a new set of architectures, known as Adapters, to tackle this problem. Adapters are low-cost plug-ins that are mounted on pre-trained models, so when adapting the model only these small sets of parameters are updated.

Bapna et al. (2019) proposed an NMT variation of Adapters. In their model, a dedicated component is added inside each layer that is responsible for transitioning in-between domains. However, all these solutions perform in centralized settings. Roosta et al. (2021) studied this problem in the context of FL and showed that Adapters might not be aligned well with the distributed nature of FL. As they reported, Adapters seem to be suitable to connect two domains but when exposed to several domains in FL, they diverge too much from their main distribution, such that using them in the body of clients drastically downgrades performance.

To address the problem, they introduced additional and dedicated layers (as opposed to intra-

layer modules in Adapters), called Controllers, that are designed to be exchanged between the server and clients. These new layers are placed in-between client model's original layers and deal with external information sent/received to/from the server. Since they only exchange Controllers, they are able to reduce the communication bandwidth. This work is the closest to ours so we use it as our main baseline.

The model proposed by Roosta et al. (2021) suffers from two issues. They randomly introduce new layers (Controllers) but there is no mechanism defined to determine the number those layers, i.e. it is not investigated that how many Controllers are required under different conditions. It is also not clear where these layers need to be placed, and it is only discovered through experimental explorations. On the contrary, we introduce a simple yet effective heuristic to select a subset of impactful parameters during communication. In our solution, we do not need to deal with the aforementioned issues.

4 Low-Cost Domain Adaptation in FL

In our FL setting, we initialize each client with a high-quality but generic NMT engine. Clients use local data to fine-tune their internal model and the combination of a pre-loaded model with local data should lead to better quality. Clients also connect with the server regularly to transfer local knowledge and contribute to the aggregation phase. In such a process, domain adaptation happens naturally. Inspired by Roosta et al. (2021), we implemented this idea and observed a substantial boost in our translation engines. Not only are we able to deliver better results but also we train NMT models in a distributed and private fashion. However, we noticed that *communication* could be quite costly.

In the default configuration, for every `Pull` (in Algorithm 1) a large NMT engine has to be be exchanged, which might not be necessary. In order to clarify why, we designed an experiment whose information is illustrated in Figure 1. In this experiment, we pick three of our engines and train them for 120K steps within the FL pipeline. For each model, we compare tensors from the 120K checkpoint to their 110K peers and measure how much they changed in-between these two checkpoints. More specifically, for a given tensor w, we compute the pair-wise difference between values from the two checkpoints ($w_d = w_{120K} - w_{110K}$), then compute the absolute-value norm of the difference tensor ($\|w_d\|$). The norm value indicates the shift of each tensor during FL rounds. Figure 1 provides a histograms of norm values that belong to different tensors from the encoders and decoders of our translation engines (for more information about the engines and datasets see Section 5).

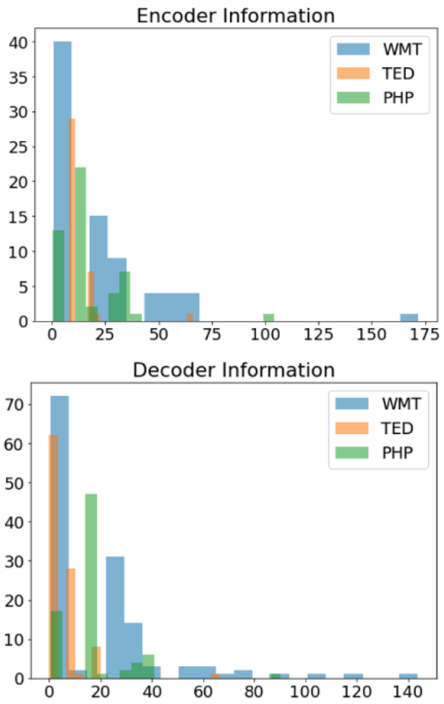

Figure 1: The histograms of the norm values of the difference tensors computed for the 110K and 120K checkpoints. The x and y axes show the *norm* and the *number of tensors*, respectively. Information related to encoder tensors is visualized in the upper half and the bottom sub-figure consists of decoder tensors' information. The first blue bar of the encoder sub-figure indicates that around 40 tensors in the WMT encoder only changed slightly from the checkpoint 110K to 120K as their norm is in the range [0,5], whereas the last bar on the other end of the same sub-figure shows that around 2 tensors in the WMT model changed drastically as the norm of their difference tensors is close to 175.

Results from Figure 1, together with our other observations, show that a small subset of tensors diverge substantially (mostly shown in the right half of the figures), but for the rest, there is a heavy concentration around small norms. More interestingly, we realized that this is a pattern that consistently occurs from one round to the next in our FL experiments, namely each tensor either belongs to a set of highly-fluctuating parameters or it only changes marginally and lies in the less active set. The fluctuation threshold can change but tensors almost always stay in their respective clusters between

rounds. We used this finding as a basis of our design to improve bandwidth consumption, such that we decided to focus on either highly fluctuating tensors or those in the other cluster and only `Pull` one type of tensors during communication. The strategy is simple but has led to promising results.

More formally, a variation of the aforementioned idea can be simply formulated as shown in Equation 2:

$$DP_g^c = \{w_r^t; \|w_r^t - w_{r-1}^t\| \geq \theta\} \qquad (2)$$

DP_g^c in the *r-th* round consists of all tensors (w_r^t) that deviate from their previous values in the previous round by θ. *DP* stands for *Dynamic Pulling* and g indicates that the difference norm of candidate tensors should be *greater than or equal* to the threshold. *DP* is exclusively computed for each client which is specified with the c superscript. Moreover, the *DP* sets for decoders and encoders are calculated separately as they vary at different scales. Based on Equation 2, we do not need to `Pull` all tensors but each client decides what to share with $\mathcal{S}$ (only highly-fluctuating tensors in this case). Figure 2 visualizes this concept.

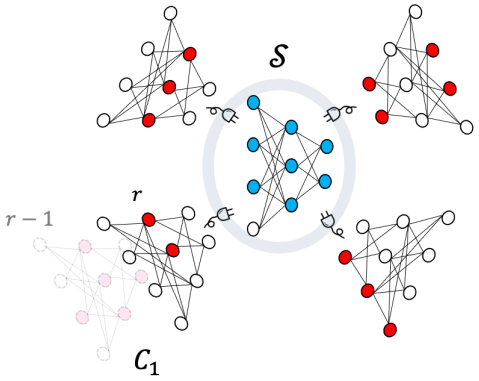

Figure 2: DP-based communication in our FL setting.

As shown in the figure, C_1 computes the difference between tensors from rounds r and $r-1$ and decides to only share two tensors (vertices in diagrams) that have the highest norms. In this scenario, the communication cost between C_1 and $\mathcal{S}$ is approximately reduced by 78% for the pulling phase as only 2 out of 9 tensors (22%) are transferred. The exact percentage of the bandwidth saved in this communication protocol directly depends on the client's architecture and θ, but the figure uses an imaginary scenario to explain the reduction mechanism.

The intuition behind DP_g^c is that only highly-fluctuating tensors should be involved in the communication process. It assumes that the main *adaptation* (or *out-domain*) knowledge lies in those tensors and what each client needs to learn about its external world is only communicated through such active tensors. Therefore, clients only need to exchange them with the server. This is an assumption that might either result in effective communication or conversely hurt the client. Because, if by any chance *local* (or *in-domain*) knowledge is stored in such tensors, DP_g^c manipulates the most important parameters and overwrites them with external domains' information. In other words, there is a possibility that the reason for observing high fluctuations in active tensors is not because they carry the community knowledge but because they are responsible to learn local data, so they have to vary frequently to adjust and learn local data.

If this second assumption is correct, modifying the active set can easily delay the convergence of local models, and thus deteriorate their quality. Due to this concern, we propose another alternative, DP_l^c, which relies on less active tensors for communications. The pulling condition for DP_l^c is as in Equation 3:

$$\|w_r^t - w_{r-1}^t\| < \theta \qquad (3)$$

which means, unlike the previous case, highly-active tensors are protected from external updates and only modified using local data. The less active tensors are assumed to be the representatives of external domains, so they are shared with the server and co-trained with other tensors. Both types of tensors contribute to the local training process, but this time the less active group is responsible for bringing the community knowledge to the client. In the next section, we compare DP_l^c and DP_g^c and show which strategy is more impactful.

5 Experimental Study

5.1 Hyper-Parameters and Datasets

Since our main baseline is the model proposed in Roosta et al. (2021), we follow their setting in the interest of fair comparisons. For our translation engines, we use Transformers with their *base* configuration (Vaswani et al., 2017). Encoders and decoders have six layers (each), attention modules have eight heads, word embeddings and internal projection layers are 512-dimensional vectors, and the inner layer in the position-wise feed-forward

Corpus	Source (De)		Target (En)		Sentences
	Words	Tokens	Words	Tokens	
WMT	119,920,225	1,529,872	126,731,132	691,150	4,468,841
OS	33,502,036	339,627	37,373,751	174,670	4,500,000
TED	2,710,904	99,221	2,861,006	45,364	143,837
PHP	322,546	13,001	318,788	8,521	39,708
UB	96,355	13,515	90,839	9,234	13,246

Table 1: Statistics of the training datasets. The second and third columns show the number of all words and unique tokens for the source and target languages, respectively, and the last column is the number of parallel sentences. *OS* is a large collection, so we randomly selected a subset of it for our experiments. As the statistics show, we have different sets with different sizes from different domains which helps us have a fair and realistic simulation.

module has 2024 dimensions. All other hyper-parameters/variables such as the training algorithm and scheduling are the same as the *base* configuration unless they are clearly indicated in the paper. We used four NVIDIA V100 GPUs for all experiments.

Similar to Roosta et al. (2021), we work on the the German→English direction. To train/test the models, we use five datasets of *WMT*,[2] *OpenSubtitle* or *OS* (Lison and Tiedemann, 2016), *PHP*, *Ubuntu* or *UB*, and *TED* (Tiedemann, 2012) where all corpora are *normalized* and *tokenized* with the scripts provided by Moses.[3] Table 1 provides the statistics of the training datasets.

For the test and development sets of the *WMT* model, we use *newstest-14* and *newstest-13*, respectively. For others, we randomly select 4,000 sentences: 2,000 for the test and 2,000 for the development sets.[4] We also pre-processed datasets to segment words into sub-words by BPE (Sennrich et al., 2016). This helps create a shared vocabulary for source and target languages of all models and avoid out-of-vocabulary entries. Our BPE setting extracts 30K unique tokens for each of the source and target sides.

One critical hyper-parameter in our model is θ. Considering the selection criterion in Equation 2, a small value of θ allows the majority of tensors to be transferred and hence leads to a minimal reduction in bandwidth. A very large value is also not plausible as it filters lots of tensors and prevents the client from receiving external knowledge. One solution is to run an exhaustive search to find the best value, which clearly is an expensive process and sometimes impossible in the case of FL. Vacillating between different options to set up θ requires the engagement of both client and server and could in fact be more costly than simply pulling all tensors. To cope with this and also make our results easily reproducible, we simply consider the median of the differences for θ, meaning we only transfer *half* (50%) of the tensors and ignore the rest. The selection criterion determines which half. In DP_g^c, we consider the active half and in DP_l^c the less active half is exchanged. Our results show that this simple strategy leads to effective communication without compromising much on quality.

5.2 Centralized Model Training

Our baseline results are summarized in Table 2. The evaluation metric used in all experiments is BLEU (Papineni et al., 2002) computed by SacreBLEU (Post, 2018).[5] As expected, models work accurately on in-domain data but perform poorly on other domains, specially if the domain is significantly different from training samples, e.g. the *PHP* model's BLEU score is *zero* when translating *WMT* test sentences.

In order to remedy the poor quality for out-domain data, we train a central model and adapt it to all other domains with two techniques of *data combination* and *chained fine-tuning*. In *data combination*, we simply concatenate all corpora to create a much larger training set. We initialize the central model with *WMT* parameters and retrain it for extra 50K steps with the new dataset. In *chained fine-tuning*, we do not combine datasets but instead fine-tune the central model sequentially using each domain's training set for additional 50K

[2] http://statmt.org/wmt14/translation-task.html

[3] https://github.com/moses-smt/mosesdecoder

[4] The same sets used in Roosta et al. (2021)

[5] https://github.com/mjpost/sacrebleu

	WMT	*OS*	*TED*	*PHP*	*UB*
WMT	**33.66**	18.57	29.22	8.04	12.41
OS	13.66	**23.58**	24.22	7.84	13.83
TED	12.09	13.59	**29.32**	6.67	10.15
PHP	0.00	0.26	0.26	**34.48**	0.00
UB	0.28	0.78	0.75	2.30	**30.15**

Table 2: Baseline results for the De→En direction. Models are trained for 100K steps. The first column indicates which training set is used to train the engine and other columns show models' performance on different test sets, e.g. [*OS*][*TED*] = 24.22 indicates the BLEU score of a model trained on the *OS* training set and tested on the *TED* test set. The best BLEU score for each test set is bold-faced.

steps (10K for each), i.e. we start from the *WMT* model, then sequentially fine-tune it over *UB*, *OS*, and other datasets one after another. This sort of fine-tuning could suffer from catastrophic forgetting, so we ran different experiments to figure out the best order of fine-tuning.

The *chained fine-tuning* strategy could provide relevant baselines for FL experiments. Imagine a scenario where the central model is shipped to a client environment and it is updated there with multiple local datasets. In *data combination*, we assume that all data is accessible at training time (a fully-observable environment with full access to all domains' data) whereas *chained fine-tuning* pictures a more realistic scenario by forcing to update the central model gradually on the client side.

Table 3 summarizes results for these two fine-tuning methods. Exposing the central model to other domains' data yields much better quality. *Data combination* clearly outperforms and it shows the impact of having direct access to data; the privilege that we do not have in settings such as *chained training* and FL.

Technique	*WMT*	*OS*	*TED*	*PHP*	*UB*
chained	18.26	**23.51**	28.19	16.14	23.05
combination	**33.50**	21.82	**31.51**	**37.56**	**35.61**

Table 3: Domain adaptation results in centralized settings.

5.3 Federated Learning Results

Models reported in the previous section (specially in Table 3) are high-quality engines that are trained in a centralized fashion and provide acceptable performance for all domains. Fine-tuning addressed the problem of poor quality for out-domain data, but as discussed previously, centralized fine-tuning and having access to out-domain data might not be always possible. Therefor, in this section we try to train comparable alternatives in an FL setting.

Our setting has one server and five clients (one for each dataset). In the interest of fair comparison between the FL and centralized approaches, we initialize all the clients with *WMT* parameters. Each client updates its model with local data and shares it with the server in each round. We `Pull` client updates after 10K steps of fine-tuning for aggregation and repeat this process for 5 rounds. In total, each model is fine-tuned for 50K steps which is identical to the setting we used for centralized training. Results for this experiment are summarized in Table 4.

	WMT	*OS*	*TED*	*PHP*	*UB*
$\mathcal{S}$	**33.97**	19.17	30.8	37.32	47.9
WMT^c	32.07	18.28	29.55	9.55	13.75
OS^c	19.05	**23.39**	27.85	13.57	18.58
TED^c	17.37	16.05	**34.30**	11.83	17.05
PHP^c	4.07	4.33	7.48	**45.07**	10.90
UB^c	0.77	4.27	5.66	14.98	**49.51**

Table 4: FL results with 1 server ($\mathcal{S}$) and 5 clients (indicated with the c superscript), e.g. OS^c is a client initialized with WMT parameters and updated with its own data (*OS* training samples).

The first row in Table 4 belongs to the server. FL affects model training quite positively and provides significantly better BLEU scores, specially for those low-performing models such as *PHP*. The average BLEU score of the server over different domains is 33.83, which is **1.83** points higher than that of the best model reported in Table 3. This means, even though FL does not access clients' data, it is more impactful in fusing information and training mixed-domain engines. This outcome for a complex task such as NMT was unexpected.

After the final FL round, the server parameters are pushed back to clients so they can also benefit from the result of aggregation. At this point, each client *can* decide to run another phase of fine-tuning with local data over the server parameters. This is a trade-off between being domain specific and remaining generic. Results for this process

are listed from the second to last rows in Table 4, e.g. *PHP*^{c} after the last `Push` step has access to all server parameters so its BLEU score on in-domain data is 37.32, similar to that of the server. It is also able to translate other domains with the average BLEU score of 32.96. However, as the fifth row shows, if it decides to fine-tune the last parameter set it received from the server with its own local data, its BLEU score increases from 37.32 to 45.07 on the in-domain test set, but at the same time it loses its generalization over other domains.

We also ran an ablation study to evaluate how the number of FL rounds impact model quality. Two important factors in FL settings are the number of clients and training rounds. We can pass over the first one as we have a cross-silo setting with a limited number of clients, but Table 5 and Figure 3 provide additional information on the second hyper-parameter.

	WMT	*OS*	*TED*	*PHP*	*UB*
$\mathcal{S}_5$	**33.97**	19.17	30.8	37.32	47.9
$\mathcal{S}_{10}$	31.90	19.63	31.04	38.33	48.63
$\mathcal{S}_{50}$	31.05	**20.83**	**32.27**	**43.99**	**51.07**

Table 5: FL results for *rounds* $\in \{5, 10, 50\}$. *rounds* = 50 means aggregation occurs 50 times between checkpoints 100K (where FL training starts) and 150K (where FL training ends).

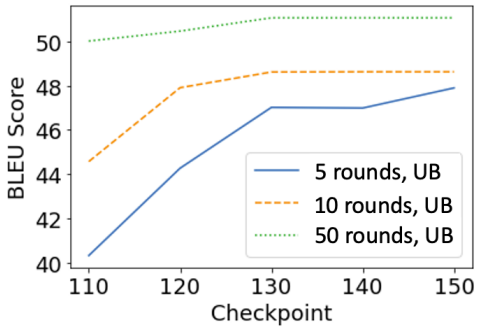

Figure 3: The learning curve of the *UB* model for *rounds* $\in \{5, 10, 50\}$.

Results from the figure/table above show that the number of rounds and model performance increase proportionally in low-quality clients such as *UB* or *PHP*. This was expected since with a higher number of rounds clients are updated more frequently with rich information from the central server. However, it comes at a price as it increases communication load. It also delays local model training, because in each round the client has to suspend training to read server values and updates its internal model. For other high-quality clients such as *WMT*, higher rounds lead to some degradation since they receive external updates from less-accurate peers and have to compensate for their low quality. The choice of the number of rounds is a trade-off between quality and bandwidth.

5.4 Dynamic Pulling Results

We proposed a novel technique for better pulling and mentioned that it is able to reduce the communication load yet maintain model quality. Table 6 reports related results to support our claims.

	WMT	*OS*	*TED*	*PHP*	*UB*
default	33.97	19.17	30.8	37.32	47.90
DP_l^c	29.28	**19.17**	**30.88**	**36.33**	**45.74**
DP_g^c	**30.74**	18.28	27.61	13.69	32.16
random	24.58	19.16	30.57	35.33	42.50

Table 6: The impact of different communication techniques on model quality. The first row is copied from Table 4 for easier comparison. The bold-faced numbers are the best results obtained by DP-based techniques.

The average BLEU scores for the *default*, DP_l^c, and DP_g^c methods on different domains are 33.83, 32.28, and 24.49, respectively. This means the assumption that less active tensors are responsible for domain adaption could be true and highly-fluctuating tensors should only be kept for learning in-domain knowledge. We also provide results from our *random* configuration, in which we exchange the same number of parameters as in DP_l^c and DP_g^c but those parameters are selected randomly. The comparison between *random* and other alternatives shows that the selection criterion directly affects model quality.

Although there is a gap by 1.55 points between *default* and DP_l^c (which is meaningful in NMT), DP_l^c could still be a strong candidate when training large models in the context of FL, because the number of parameters exchanged in each pulling step is *45,724,160* for *default* whereas this number is only *22,863,104* (50% less) for DP_l^c. It should also be noted that pulling occurs not once but for multiple rounds and saving 50% each time is a significant gain. Moreover, DP_l^c still performs on par with *data combination* which is a strong but centralized and not private baseline.

5.5 Comparison to Controllers

As previously discussed in Section 3.1, Controllers (Roosta et al., 2021) only exchange 4 layers (2 from the encoder and 2 from the decoder in a 12-layer Transformer) with the server in the `Push` and `Pull` phases, which means the communication bandwidth they reduce is around 66% ($\frac{4}{12} \approx 33\%$ of the layers are only exchanged between the server and clients). In our case, *DP* only affects `Pull` which leads to 25% bandwidth reduction in client-to-server exchanges.[6]

We challenged our model to see if we can also save 66% in bandwidth by sending/receiving the same number of parameters as in Controllers. More precisely, we applied our threshold-based strategy in *both* `Push` and `Pull` and modified the value of the threshold such that it only accepts 33% of the parameters to exchange with the server. Table 7 summarizes results of this experiment.

	WMT	*OS*	*TED*	*PHP*	*UB*
DP_l^c	29.52	19.25	**31.53**	**36.19**	**45.40**
DP_g^c	29.09	19.07	30.93	34.47	41.57
6E6D (0-3)	31.13	19.19	30.95	33.79	32.85
8E8D (0-6)	**31.79**	**20.02**	30.6	32.43	33.41

Table 7: Dynamic parameter selection versus Controllers.

In our comparison, we selected the two best performing Controller models reported in Roosta et al. (2021). The *6E6D (0-3)* configuration is a Transformer with a 6-layer encoder and 6-layer decoder whose first and fourth layers are selected to act as Controllers. In the *8E8D (0-6)* configuration, instead of using the original encoder/decode layers as Controllers four additional layers (two for the encoder and two for the decoder) are defined which are placed after layers 0 and 6. This means, in an 8-layer encoder/decoder the first and seventh layers are Controllers and the rest are ordinary layers.

Results show that our model could be a reliable alternative for communication-efficient FL, even though we aggressively limited the number of parameters exchanged in this new setting. Moreover, in our model we do not need to define additional layers. Unlike Controllers, we also do not have to deal with finding the correct position to place Controller layers. According to Roosta et al. (2021), the final model performance is directly impacted by misplacing Controllers and our solution solves that problem.

[6] `Pull` is only 50% of the communication where we excluding half the parameters using the threshold, so $50\% \times 50\% = 25\%$.

6 Conclusion and Future Work

In this paper, we reported a set of benchmarking result for NMT in the context of FL. We also proposed an effective technique to reduce the communication bandwidth. Our solution tries to determine a subset of parameters that are responsible for learning out-domain knowledge and only exchanges them with the server.

In future work, we are interested in *i*) adding more languages to train multilingual engines, *ii*) improving communication protocols even further, *iii*) comparing other FL algorithms in the presence of *DP*, and *iv*) investigating NMT in cross-device settings.

Acknowledgments

We would like to thank our anonymous reviewers for their valuable comments as well as Liling Tan, Grant Strimel, and Sriram Venkatapathy (from Amazon) for providing feedback on the first version of this paper.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*.

Duc Bui, Kshitiz Malik, Jack Goetz, Honglei Liu, Seungwhan Moon, Anuj Kumar, and Kang G Shin. 2019. Federated user representation learning. *arXiv preprint arXiv:1909.12535*.

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

Chenhui Chu and Rui Wang. 2018. A survey of domain adaptation for neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1304–1319, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep

bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Suyu Ge, Fangzhao Wu, Chuhan Wu, Tao Qi, Yongfeng Huang, and Xing Xie. 2020. Fedner: Privacy-preserving medical named entity recognition with federated learning. *arXiv e-prints*, pages arXiv–2003.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR.

Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*.

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

Andrew Hard, K. Rao, Rajiv Mathews, F. Beaufays, S. Augenstein, Hubert Eichner, Chloé Kiddon, and D. Ramage. 2018. Federated learning for mobile keyboard prediction. *ArXiv*, abs/1811.03604.

Corentin Hardy, Erwan Le Merrer, and Bruno Sericola. 2019. Md-gan: Multi-discriminator generative adversarial networks for distributed datasets. In *2019 IEEE international parallel and distributed processing symposium (IPDPS)*, pages 866–877. IEEE.

Florian Hartmann, Sunah Suh, Arkadiusz Komarzewski, Tim D Smith, and Ilana Segall. 2019. Federated learning for ranking browser history suggestions. *arXiv preprint arXiv:1911.11807*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Shaoxiong Ji, Wenqi Jiang, Anwar Walid, and Xue Li. 2020. Dynamic sampling and selective masking for communication-efficient federated learning. *arXiv preprint arXiv:2003.09603*.

Shaoxiong Ji, Shirui Pan, Guodong Long, Xue Li, Jing Jiang, and Zi Huang. 2019. Learning private neural language modeling with attentive aggregation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Heejae Kim, Taewoo Kim, and Chan-Hyun Youn. 2020. On federated learning of deep networks from non-{iid} data: Parameter divergence and the effects of hyperparametric methods.

David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. 2019. Federated learning for keyword spotting. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6341–6345. IEEE.

Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. 2021. Federated learning on non-iid data silos: An experimental study. *arXiv preprint arXiv:2102.02079*.

Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. 2019. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *arXiv preprint arXiv:1907.09693*.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2018. Deep gradient compression: Reducing the communication bandwidth for distributed training. *ICLR*.

Pierre Lison and Jörg Tiedemann. 2016. OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*.

Dianbo Liu and Tim Miller. 2020. Federated pretraining and fine tuning of bert using clinical notes from multiple silos. *arXiv preprint arXiv:2002.08562*.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Peyman Passban, Puneeth SM Saladi, and Qun Liu. 2020. Revisiting robust neural machine translation: A transformer case study. *arXiv preprint arXiv:2012.15710*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on*

Machine Translation: Research Papers, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Hanchi Ren, J. Deng, and Xianghua Xie. 2021. Grnn: Generative regression neural network – a data leakage attack for federated learning.

Tanya Roosta, Peyman Passban, and Ankit R. Chadha. 2021. Communication-efficient federated learning for neural machine translation. *ArXiv*, abs/2112.06135.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. *arXiv preprint arXiv:2010.11918*.

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.

Joel Stremmel and Arjun Singh. 2020. Pretraining federated text models for next word prediction. *arXiv e-prints*, pages arXiv–2005.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Su Wang, Mengyuan Lee, Seyyedali Hosseinalipour, Roberto Morabito, Mung Chiang, and Christopher G Brinton. 2021. Device sampling for heterogeneous federated learning: Theory, algorithms, and implementation. *arXiv preprint arXiv:2101.00787*.

Xing Wu, Zhaowang Liang, and Jianjia Wang. 2020. Fedmed: A federated learning framework for language modeling. *Sensors*, 20(14):4048.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19.

Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. 2020. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*.