CS4290 Project Report
Dynamic VF-PS: Extending Participant Selection in Vertical Federated Learning to Dynamic Settings via Data Padding and Random Picking

Roberto Gheda
5863503

Abstract

VF-PS paper introduces the problem of *Vertically Federated Participant Selection*. However, it provides a solution that only works in *static settings*. That are, settings in which data distributions don't change over time.
This report introduces *Dymamic VF-PS*: an implementation of VF-PS for dynamic settings based on *data padding*. Furthermore, it introduces *Random Picking* as a a scalable, secure and easy to implement solution to the VF-PS problem in both the static and the dynamic setting.
In the appendix an implementation of an Online MI Estimation algorithm based on Random Picking is shown and tested.

Original paper: *"VF-PS: How to Select Important Participants in Vertical Federated Learning, Efficiently and Securely?"* presented at *36th Conference on Neural Information Processing Systems* by Jiang et al.

1 Introduction

Vertical Federated Learning (VFL) is a distributed machine learning paradigm that involves a consortium of m participants $\mathcal{P}\{P_1, ..., P_m\}$ trying to learn a shared model. The data is *vertically partitioned* between participants; meaning, each of them owns a disjoint subset of features associated with the same data samples. The goal of VFL is to learn a shared ML model over the joint feature space, in a secure and privacy-preserving way. In order to achieve this, VFL systems usually exploit computationally expensive methods such as *Homomorphic Encryption* or *Secure Multi-Party Computation*. These methods' computational costs make VFL systems scale poorly with respect to the number of clients.

Another big concern with VFL systems scalability is *diminishing return*. Meaning, when the number of clients increases, these new clients may not bring a meaningful quality improvement to the model while making the system slower.

1.1 Report contribution

VF-PS currently requires to run a *participant selection* algorithm before the training and to keep said selection unchanged. That means, it is not designed for dealing with scenarios in which data distribution changes overtime. We call such scenarios *dynamic settings* (similarly, we refer to standard VFL problems as *static*).

This work introduces *Dynamic VF-PS*: an implementation of VF-PS for dealing with dynamic settings based on *data padding*. Then, introduces *Random Picking* strategy as scalable participant selection algorithm.

This report answers the following research questions:

- *How do different data padding strategies influence the learning process?*
- *How does VF-PS algorithm scale in a dynamic setting?*
- *How does Random Picking compare to VF-PS in dynamic settings?*

This report contributions are:

- *Dynamic VF-PS*: an implementation of VF-PS for dealing with dynamic settings based on *data padding*.
- Comparison between different data padding strategies performance on random picking.
- Experiments on Random Picking behavior in the dynamic setting.
- A brief analysis on VF-PS scalability issues. n addition, an implementation of Online MI Estimation via Random Picking is provided in Appendix B. Dynamic VF-PS' code and experiments are provided in a GitHub repository[1].

2 Background: Vertical Federated Participant Selection

The original VF-PS paper[2] introduces an algorithm for VFL systems to select l out of m participants that are *most important*. The remaining participants are, then, not used in the training process.
VF-PS is clearly related with the field of feature selection. However, it manages to impose new challenges. In fact, VF-PS should provide *security* and *privacy guarantees*, and, furthermore, be *more efficient* than training a single VFL model over all participants.

2.1 VF-PS algorithm

VF-PS algorithm is split in two phases: *Group Testing* and *MI Estimation*. During MI Estimation, VF-PS tests a subset of k out of m clients and gives it a *score* that is, ideally, proportional to its features' importance. During Group Testing, instead,

VF-PS generates said clients' subsets and runs MI Estimation on them. Each client is then given a score based on results obtained when it took part of the MI estimation process. The l highest rating clients are, eventually, selected.

2.1.1 MI Estimation

The paper provides two KNN-based MI Estimation implementations: Baseline and Fagin-Inspired.

On the one hand, the *Baseline Method* requires every client to send the server their *local distances*. Distances are then aggregated by the server to form global distances between samples. Eventually, a score is computed as:

$$\frac{1}{|Q|}\sum_{q}(\psi(N)-\psi(N_q)+\psi(K)-\psi(m_q)) \tag{1}$$

That, intuitively, is proportional to the sum over every data sample of the number of objects of class non-q nearer to the data sample than its k-nearest neighbor of class q in their local feature space. Here q is the class of the data sample and ψ is the digamma function $\psi(x)=\frac{d}{dx}\Gamma(x)\simeq \ln x-\frac{1}{2x}$.
All communications are encrypted using *Homomorphic Encryption*. Keys are provided by a dedicated server.

The *Fagin-Inspired Method*, on the other hand, diminishes Baseline Method's time cost by exploiting Fagin's algorithm to do a part of computation in parallel. In this version of the MI Estimation algorithm, clients are now responsible for computing local scores.

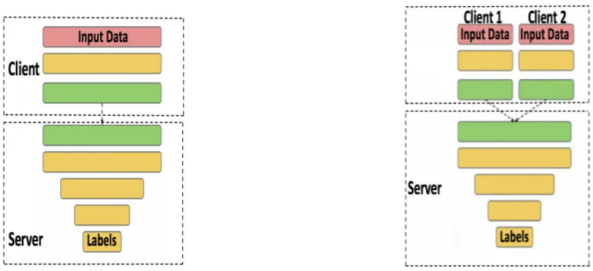
2.1.2 Group Testing

Group Testing is done by naively creating random subsets of participants to test. Each client is given a score that is the average score of all tests it has been participating in. An improved version of Group Testing that exploits Fagin's inclusion property for doing *batching optimization*, reducing the number of times the estimation algorithm has to be executed, is also presented.

2.2 Limitations

The paper provides a brief analysis of VF-PS limitations. In summary, these are:

- *Multiple training stages*; VF-PS isn't designed to work on systems in which the importance of the participants can change over time.
- *Reward assessment*; different participants may hold a different importance during training phase, which may lead to fairness problems.
- *Generalization of MI estimator*; currently, the MI estimator is only proved to work with monotonic aggregation functions.
- *Large-scale federated network*; VF-PS performance is expected to degrade when the number of participants increases.



(a) SplitNN PyVertical code (b) SplitNN VF-PS

Fig. 1: PyVertical's and Dynamic VF-PS SplitNNs comparison

As already mentioned, this work will focus on the first limitation. However, issues with MI Estimation scalability manifested during evaluation and have, therefore, been analyzed as well.

2.3 Artifact

There is no code implementation provided in the paper nor in its appendix. A code implementation of VF-PS can be found on the openreview.net web page dedicated to this paper[3]. Under the section *revisions* it is possible to download a .zip folder containing an implementation of VF-PS as *supplementary material*. However, this code is not documented nor commented. Moreover, most files and directories have cryptic names, rendering the understanding the code challenging.

For these reasons, and because the Dynamic VF-PS work would require to edit fundamental parts of the framework anyway, I decided not to use this code.

3 Framework

After an initial -failed- effort to work on FLTK, I had to move to a new framework that supports VFL. Subsequently, based on a suggestion from one of the teaching assistants, I started working with PyVertical[4][5].

3.1 PyVertical

PyVertical is a framework developed by OpenMined for training models on vertically partitioned data using SplitNNs. PyVertical exploits the PySyft library and allows to allocate two virtual workers: one server, owning all the labels, and one client, owning all the data.
This structure is different from the one needed to implement VF-PS. In order to do so, a SplitNN with multiple workers owning different vertical partitions of the data shall be built and trained.

3.2 Dynamic-VF Framework

Eventually, I ended up getting rid of most of the original PyVertical code. What is provided in the repository is a novel PySyft-based framework for training VFL systems in Dynamic settings[1].
The repository code consists of two SplitNN classes (one discrete, `discrete_splitnn.py`, one relaxed, `relaxed_splitnn.py`), responsible for handling training algorithm and participant selection, and two dataloader classes (one discrete, `discrete_distribute_dataloader.py`, one relaxed, `relaxed_distribute_dataloader.py`), responsible for handling and maintaining the dynamic data. Moreover, two example notebooks, in the `examples` directory, and two testing scripts, in the `scripts` directory, are provided.
The final core work consists of more than 1000 lines of core code (excluding examples and testing scripts).

3.2.1 Discrete Dynamic Setting

In the discrete dynamic setting, the available data completely changes every few epochs. Codewise, the `subdata` is created by picking every datapoint with a certain probability. Then, for every epoch, the `subdata` is regenerated from scratch with a certain probability.

3.2.2 Relaxed Dynamic Setting

In the relaxed scenario, the available data gradually changes overtime. Codewise, the starting `subdata` is created in the same way as in the discrete scenario. Then, at every selected datapoint, a random lifespan value is given. This lifespan value is decreased at every epoch. When it reaches zero, the datapoint is removed from the `subdata`. In the meantime, at every round any datapoint has a certain probability to be selected.

3.2.3 Data partitioning

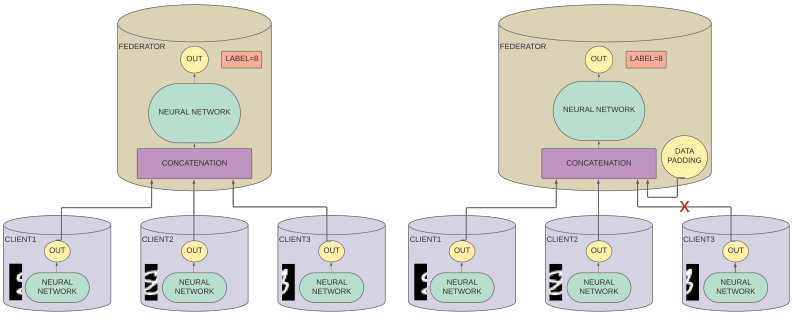
The dataset is treated as a list of tuples containing:

- `id`: identifier of the datapoint
- `data_ptr`: a dictionary of pointers to clients' private partitions of the datapoint
- `target`: a pointer to server's private partition (i.e., datapoint's target)

The data handler object has an attribute called `distributed_subdata` that represents the current state of the distributed data. That is, de facto, a subset list of the full dataset list. When the SplitNN object selects some participants, it copies such subdata and removes the non-selected owners' keys from the `data_ptr` dictionary.

3.3 PSI and Homomorphic Encryption

Both *Private Set Interction* and *Homomorphic Encyrption* are not implemented due to lack of time at disposal. I expect a PSI approach to add linear time complexity with respect to the dataset size[6][7].
Homomorphic Encryption is required for privacy constraints during the MI Estimation



(a) The SplitNN initially receives vertically partitioned data from all available clients.

(b) When the best two clients are selected, the missing data is replaced by a data padding function.

Fig. 2: Dynamic SplitNN design

procedure. Howerver, since experiments of this report will mainly focus on a random selection approach, its absence doesn't affect this report's results and conclusions.

4 Improvements

In the created framework, a SplitNN using VF-PS *Baseline KNN MI Estimation* from the original paper has been implemented. In addition, the subsequent improvements have been implemented.

4.1 Dynamic client selection for KNN MI Estimation

In Dynamic VF-PS, the federator runs a group testing algorithm that implements the baseline KNN MI estimator algorithm[2] which is responsible for selecting l out of m clients. In the original paper, non-selected clients where not included in the learning process. In a dynamic setting this is not possible, as it would require to change the whole model everytime selected clients change.
Instead, when a data sample has to be predicted, the selected clients do local computation and send them to the server. Then, the federator replaces local computation values of non-selected clients with a predefined data padding strategy.

4.2 Data Padding

In Dynamic VF-PF implementation, 4 different data padding strategies are provided:

- *Zero padding*; a naive padding solution where all missing values are replaced by zeros.
- *Ones padding*; where all missing values are replaced by ones.
- *Mean value padding*; in which each missing value is replaced by its own average value in the past.

- *Latest value padding*; in which each missing value is replaced by its latest value received.
- *Exponential weighted moving average padding*; in which each missing value is replaced by a weighted average of its past values, giving a higher weight to most recent values.

Further data padding strategies can be thought of, including machine learning strategies such as *K-Nearest Neighbours* or *Masked Autoencoders*[8] [9]. These solutions have been used for missing data generation with very good results[10] [11]. However, since the input is given from clients' neural network whose behavior changes during training, these machine learning techniques would require to learn over distributions that can drastically change over time.
In addition, training a machine learning model would further increase the system complexity, making it harder to learn the distributed one.
Finally, Masked Autoencoders are currently proven to work only on few domains in which features are strictly related to one another, like language models or computer vision[10]. Hence, they may not fit generic learning domains.

4.3 Relaxed MI Estimation

In the Discrete setting, Group Testing is ran every time the data changes. Since in the Relaxed setting data changes continuously, this would be a great bottleneck for performance. Therefore, when data owned by clients evolves, the server updates participants' scores in a pessimistic fashion. This means that selected clients' scores decrease following the worst case scenario of evolution of equation 1 while non-selected ones increase following the best case scenario.
As soon as optimal selection may have changed, Group Testing is ran from scratch.

4.4 Scaling down MI Estimation

A VF-PS issue that isn't discussed in the original paper is *MI Estimation scalability with respect to dataset size*. When dataset is large, running KNN for every data sample contributes to tighten the already problematic performance bottleneck caused by the repeated execution of Group Testing. Therefore, scaling down MI Estimation is drastically needed to make Dynamic VF-PS feasible.
In Dynamic VF-PS a naive solution of randomly using a subset of the available dataset is provided. Further ideas on how to improve this point are provided in the Future Work section.

It is worth of notice that, despite the fact that Communication Batching improves scalability with respect to the number of clients, it does not solve the aforementioned problem. In Figure 3 it is shown that VF-PS can require up to 10k seconds (i.e, 20 minutes) for running participant selection.

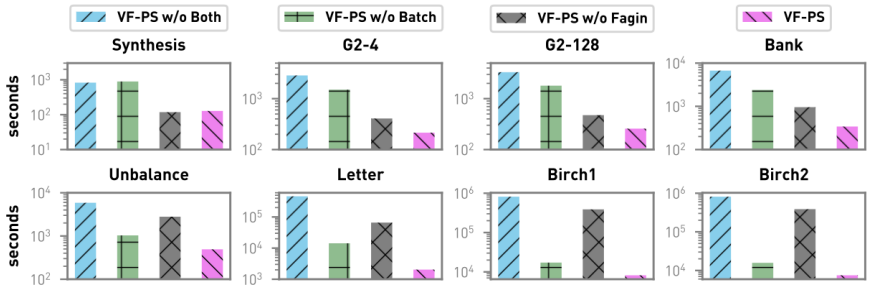


Fig. 3: VF-PS' ablation study (Figure 6 from the original paper)

4.4.1 Random Picking

As a naive but efficient solution to this issue, a *Random Picking* strategy is provided and tested. Despite its simplicity, experiments show interesting results while providing a scalable approach to the problem.

Furthermore, Random Picking doesn't require any additional communication between participants, making it a secure and privacy-safe solution for VFL systems.

5 Experiments

Due to inability to access cloud resources (i.e., Google Cloud removed my credits for unspecified reason, Kaggle notebook only runs Python3.10 while my framework works on Python3.7), all experiments are conducted on a local Docker image. This prevented doing long experiments with a high number of clients and large datasets. Moreover, KNN MI Estimator scalability issues made its testing unfeasible in this setting.

All experiments are conducted with 5 participants (1 server, 4 clients) trying to learn a shared model on MNIST[12]. Clients' local neural networks are composed by an input layer of size 28×7, one layer of size 128 and one output layer of size 64. Server aggregates clients' local output to form an input layer of size 256, then it passes them through a layer of size 128 and, finally to a output layer of size 10. In all tests, except where specified, 2 out of 4 clients are randomly selected for training.
Except where specified, experiments are done in the discrete dynamic setting with frequency of dataset update of 20%.

5.1 Random Picking convergence

In Figure 4 and Table 1, random picking performance -similarly to what did in Table 2 of the original paper[2]- is compared with the brute-force best worst client selection. Here we notice that random pick test loss convergence closely follows the two aforementioned methods' ones.

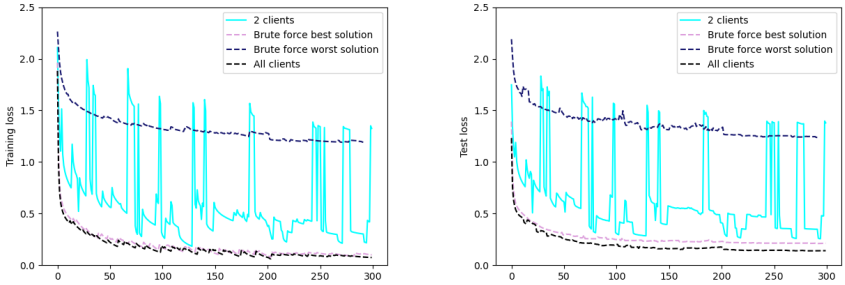


Fig. 4: Comparison of loss in random picking with brute-force best choice and brute-force worst choice when selecting 2 clients.

	Train loss	Test loss	Train accuracy	Test accuracy
1 client worst brute-force	1.469	1.494	23.0%	22.4%
1 client random picking	0.436	0.47	64.7%	62.5%
1 client best brute-force	0.289	0.358	74.9%	69.9%
2 client worst brute-force	1.191	1.238	30.4%	29.0%
2 clients random picking	0.183	0.255	83.3%	77.5%
2 clients best brute-force	0.078	0.21	92.5%	81.1%
3 client worst brute-force	0.273	0.334	76.1%	71.6%
3 clients random picking	0.113	0.169	89.3%	84.5%
3 clients best brute-force	0.083	0.205	92.2%	81.1%
Training with all clients	0.062	0.139	94.0%	87.1%

Table 1: Losses for different numbers of selected clients after 400 epochs. Reported values are maximum so far.

These 'unstable events' happen when the federator changes its selection from an optimal to non-optimal one.

5.1.1 Study on convergence when number of clients changes

In Table 1, when the number of participants to be selected is small, the model's performance degrades due to the diminishment of available input data. However, random picking's test loss is relatively close to the brute force best choice in all tests.

As shown in Table 1, Random Picking, compared to the best brute-force solution, loses around 2.9% in training accuracy when choosing 3 clients, 8.8% when choosing 2, 9.8% when choosing 1. My intuition behind these numbers is related to probability of picking the optimal clients combination. We are from now on assuming that clients owning central parts of the images are the best choice. This was empirically shown during experiments ran to compute brute-force solutions as well as in experiments in Appendix B.

Firstly, when choosing 3 clients we have 50% chance of picking both and 50% chance

	Train loss	Test loss	Train accuracy	Test accuracy
Zero padding	0.261	0.298	77.0%	74.3%
Ones padding	2.302	2.302	10.0%	10.0%
Latest value padding	0.371	0.484	69.0%	61.6%
Mean value padding	0.355	0.355	70.1%	70.1%
Moving average padding	0.312	0.371	73.2%	69.0%

Table 2: Different techniques losses after 200 epochs. Reported values are maximum so far.

of picking one of the two. This makes this scenario highly reliable since we always train on at least one of the two clients that take part in the optimal combination. Moreover, 50% of the times, we train on both the central clients.
Then, when choosing 2 clients, we have chance 1/6 of picking both, 4/6 of picking one of them, and 1/6 to pick none.
Lastly, when choosing 1 client, we have 50% chance of picking one -optimal choice- and 50% of not picking a central one. However, since we can only use one of the two central clients, our real probability of training on the optimal solution is 25%.

I expect these probability concepts to be the backbone of future research on Random Picking's theoretical foundations and to prove a bound on random picking's delay on convergence.

It is worth of notice that, due to computational limitations, not all the tests have ran for the same number of epochs. No test has crashed before 300 rounds. More details about this are available on the experiment notebook in the project repository (`experiments/extra_week/m_out_of_n_clients.ipynb`).

5.2 Data padding strategies comparison

Different padding strategies have been implemented and evaluated in Table 2. Zero-padding is the best choice since it converges faster and is more stable.
Further details about the intuition on this are provided in Appendix A. Losses convergence of these experiments are provided in Appendix C.2.

5.3 Study on subset update frequency

A comparison of convergence of the model with different update frequency values is provided in Figure 5. As expected, when the frequency value is higher, training loss and testing loss are more unstable. Even so, this doesn't seem to significantly affect convergence of test loss.

6 Future work

6.1 Further experiments

Doing further experiments on this topic is needed in order to have a full comprehension of the Random Picking algorithm behavior.

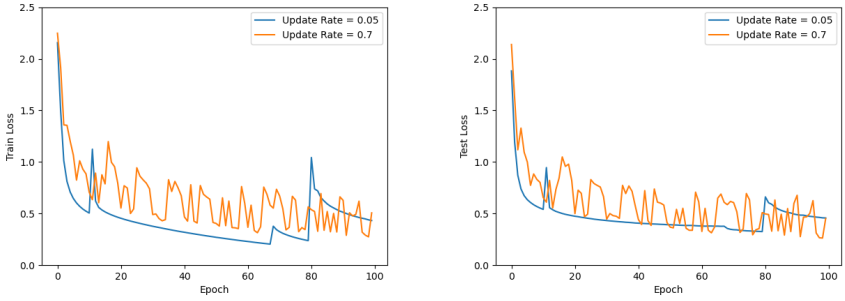


Fig. 5: Comparison of convergence with different dataset update frequencies.

Currently, Dynamic VF-PS has been tested on one dataset only and with a small number of clients. Moreover, due to the limited computation capability of my local machine, it was not possible to complete acceptable experiments in the relaxed setting, with the original VF-PS MI Estimator and with the subsampled KNN MI Estimator.

6.2 Theoretical studies on Random Picking

In section 5.1.1 some ideas to start a theoretical study have been provided. A formal proof on Random Picking learning behavior would make it a more appealing approach for research in this field.

6.3 Scaling down MI Estimation

As previously discussed, KNN MI Estimation scales poorly even in the original VF-PS scenario. Solving this issue would bring a large improvement on both the static and the dynamic setting.
Here are some proposals for future research in this field.

6.3.1 Parzen MI Estimator

Currently, VF-PS runs KNN on all datapoints to estimate MI values. In Dynamic VF-PS this issue is naively solved by running KNN on a subsample of the full dataset. In the original VF-PS paper, KNN is used to build clusters around datasamples and evaluate them based on how many samples of a different class are present in such clusters. In a similar fashion, MI values could be computed using a Parzen window instead.

Compared to KNN, Parzen is:

- *More tunable*; considering that in the Fagin version of the MI Estimator scores are computed locally, we can access a lot of important information that we are not using in KNN, like what are the most dense and sparse regions of the feature space.

- *More scalable*; as it doesn't require to compute such clusters for every datapoint but only for a few clusters centers.

Moreover, it must be taken into account that KNN requires to choose a value k for which there is no known rule of thumb. Several researchers in the past have struggled with this[13]. Some examples of KNN behavior when an improper value of k is chosen are provided in Figure 15 (Appendix D).
Parzen-based MI estimator have been used in literature for a long time[14].

6.3.2 Online MI Estimator via Random Picking

Experiments have shown that Random Picking has an unstable training loss that mainly depends on the quality of the selected clients. Random Picking could be used during the training to learn what participants to select analyzing its training loss at runtime.
In the dynamic setting, this approach would be used to avoid running Group Testing everytime client's mutual information may have changed.
In the static one, using this approach would save a lot of time before starting the training process. It needs to be highlighted that original VF-PS algorithm can take up to 20 minutes of pre-processing before starting the training procedure.

In Appendix B an implementation of Online MI Estimation via Random Picking for static settings is provided with pseudocode and some experiments. Moreover, a proposal on how to implement a dynamic setting version of the algorithm is also provided.

7 Conclusions

7.1 Answers to research questions

- *How do different data padding strategies influence the training process?* Zero-padding is the padding strategy that performs the best. An in-depth explanation on this result is provided in Appendix A.
- *How does VF-PS algorithm scale in a dynamic setting?* Running VF-PS in a dynamic setting is currently unfeasible due to its scalability issues. Even though it hasn't been tested, random subsampling can, in theory, curb this problem. In theory, Random Picking scales better than VF-PS with random subsampling anyway. More solutions to this issue have been provided in the Future Work chapter.
- *How does Random Picking compare to VF-PS in dynamic settings?* Random Picking performs close to optimal when selecting a reasonable amount of clients. When the number of clients to select is much smaller than the total, Random Picking wastes more time training on clients that are not part of the optimal solution. An Online MI Estimation technique can be implemented to solve this issue (Appendix B).

KNN MI Estimation as implemented in the orginal VF-PS paper scales poorly and cannot be implemented in a dynamic scenario.

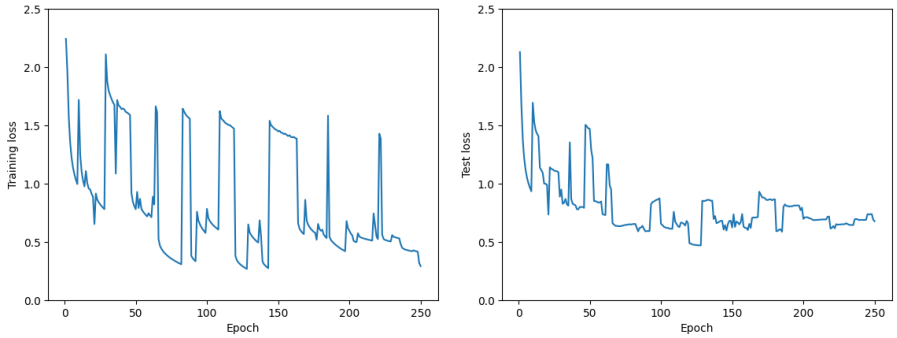


Fig. 6: Training loss and test loss when the neural network tests using all clients.

Random Picking is an algorithm that:

- Is *easy to implement*
- Has *no scalability issues*
- Requires *no additional communications* to the ones needed for training. This makes it *fast* and *privacy-secure*.

In addition, Random Picking solves fairness issues that has been brought up in several ethics reviews on the original paper[3].
Finally, Random Picking opens the way to a set of stochastic solutions that need to be explored. One of these is provided in Appendix B.

7.2 Other works on this topic

Two papers have cited the original work. None of them addresses neither the topic of dynamic settings, nor the topic of participant selection algorithms.

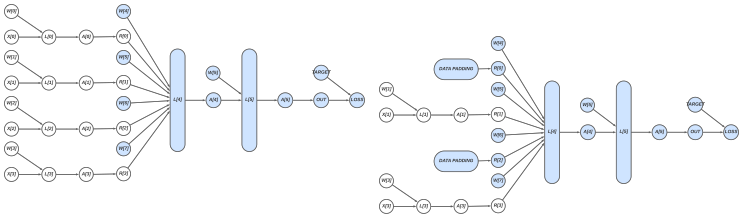
A Reasoning behind data padding

In deep learning it's perfectly safe to substitute missing values with constants. The network will learn that that constant value means 'missing' [15].
However, what we are doing is masking a large part of the input variables. Therefore, what we are doing is learning over a 'padded distribution'. This is proved in Figure 6, where the network is tested using input from all clients. It is clear that, while train loss converges to a low value, test loss doesn't.

A.1 Why does zero padding perform better?

Now the question might be: *Why does zero padding perform better than other padding strategies?*

Firstly, latest-value padding doesn't provide a constant input to train the network on



(a) Computational graph when selecting all clients.

(b) Computational graph when selecting Client 1 and Client 3.

Fig. 7: Computational graph of a SplitNN with 4 different clients. Nodes colored in blue are located server-side.

and won't bring the network loss to convergence (Figure 12).

Then, in order to understand why it performs better than other techniques like mean-value padding (in which the input value will eventually converge to a constant), we have to delve into properties of zero-padding in the forward pass and in the backpropagation.

A.1.1 Forward pass

When performing forward pass from the padded input $\mathbf{xW}$, the result will always be a tensor of zeros. This means that weights from the padded input will not affect further layers. In other words, we don't need to learn $\mathbf{W}$ that will both predict outputs when the client is selected and when it is not, making it a suitable choice for dynamic settings.

A.1.2 Backpropagation

When running the backpropagation algorithm to compute $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ we need to follow the computational graph in Figure 7.

In the scenario in Figure 7b, when deriving $\frac{\partial \mathcal{L}}{\partial \mathbf{W}[4]}$ we get:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}[4]} = \frac{\partial \mathcal{L}}{\partial L[4]} \frac{\partial L[4]}{\partial \mathbf{W}[4]} = \frac{\partial \mathcal{L}}{\partial L[4]} \mathbf{R}[4] = \mathbf{0}$$

This can be easily proven for every non-selected client $c \in C$.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_c} = \mathbf{0} \; \forall c \in \{\mathcal{C} \setminus \mathcal{S}\}$$

Hence, when using zero-padding, weights coming from non-selected clients will not be updated. Furthermore, weights update of selected clients are not influenced by padding coming from non-selected ones.

	#Times selected	Mean score	Score standard deviation	Highest score	Lowest score
Client 1	0	1.804	0.0016	1.903	1.71
Client 2	100	1.23	0.0069	1.473	1.028
Client 3	100	1.264	0.0053	1.446	1.106
Client 4	0	1.795	0.0017	1.895	1.705

Table 3: Statistics from Online MI Estimation proof of concept.

B Online MI Estimation via Random Picking

In Online MI Estimation, the goal of the algorithm is to select k participants at runtime, possibly in a communication-efficient and privacy-secure fashion. As a proof of concept of Random Picking potentialities, here I am presenting a draft implementation of Online MI Estimation for static settings and a proposal on how to implement it in dynamic settings.

B.1 Static settings

In the static setting, we need to select k participants only once. In order to achieve this, we can exploit Random Picking instability to gather information about clients mutual information. Every client is given a score based on the average training loss performed when selected. Then, the k clients with the lowest scores are selected. The algorithm is split into two phases.

In the *preprocessing phase*, we train the model using random picking as presented in the report for a few epochs. The point of this phase is to reduce random picking unfairness. In fact, if a client is selected in the first epoch, it will receive a high training loss regardless of its actual amount of mutual information.

In the *learning phase*, we actually compute the clients' scores. At every epoch, for a predefined amount of epochs, we randomly select k participants and compute the models' training loss.
Eventually, for each client, its score is computed as the average training loss performed. The k clients with the lowest score are selected.

A pseudocode of the Online MI Estimation algorithm is provided in Algorithm 1.
Statistics of its performance are provided in Table 3. The algorithm has been ran 100 times on the MNIST dataset with 5 preprocessing epochs and 5 learning epochs, managing to select the optimal clients all of the 100 times.
These results can be explained by the fact that MI distribution among clients is highly biased towards the central ones.

B.2 Dynamic settings

In dynamic settings, we need to be able to change our selection multiple times. In order to achieve this, we can randomly select k participants and change them with a probability that is inversely proportional to their training loss.
For instance, using negative log likelihood, we could provide a probability of changing

Algorithm 1 Online MI Estimation via Random Picking for Static settings

Require:
LEARNING_EPOCHS > 0
PREPROCESSING_EPOCHS > 0
Owners $\neq \emptyset$
k > 0

Ensure: Selects k owners with the highest estimated mutual information while training the model.

```
for i in 1...PREPROCESSING_EPOCHS do
    train()
end for
selected = ∅
losses[owner] = ∅ ∀ owner ∈ Owners
for i in 1...LEARNING_EPOCHS do
    selected = random_picking()
    train()
    loss = compute_train_loss()
    for owner ∈ selected do
        losses[owner] = losses[owner] ∪ loss
    end for
end for
scores = {mean(losses[owner]) ∀ owner ∈ Owners}
selected = lowest(k, scores)
```

participants of:

$$p(x) = (1 - e^{-x})^2$$

Where x is the current training loss.

B.3 Comments on Online MI Estimation

Online MI Estimation is an algorithm that can improve Random Picking performance when number of clients to be chosen is significantly low.
Compared to VF-PS, Online MI Estimation is more efficient and secure. Hence, I expect it to outperform VF-PS in both the static and the dynamic setting.
I don't see any advantage on using VF-PS instead of Online MI Estimation.

C More experiments figures

C.1 Convergence with different numbers of clients

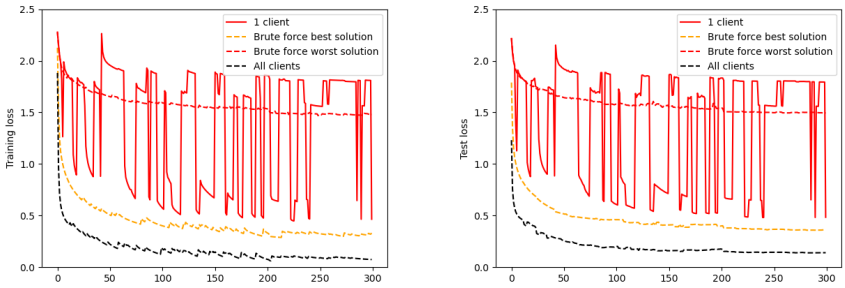


Fig. 8: Comparison of loss in random picking with brute-force best choice and brute-force worst choice when selecting 1 client.

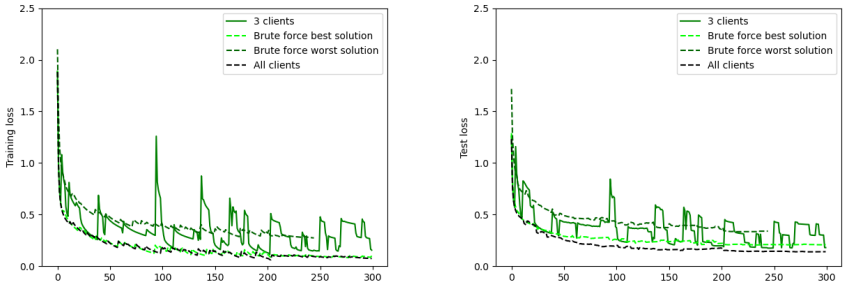


Fig. 9: Comparison of loss in random picking with brute-force best choice and brute-force worst choice when selecting 3 clients.

C.2 Data padding strategies

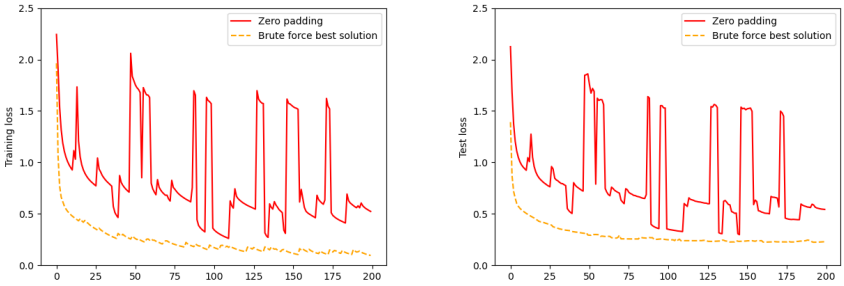


Fig. 10: Performance on zero padding.

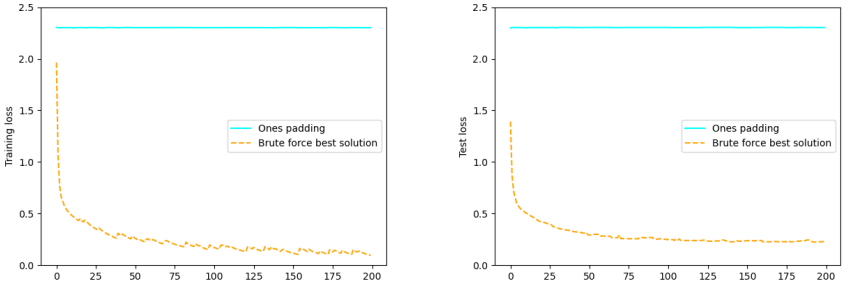


Fig. 11: Performance on ones padding.

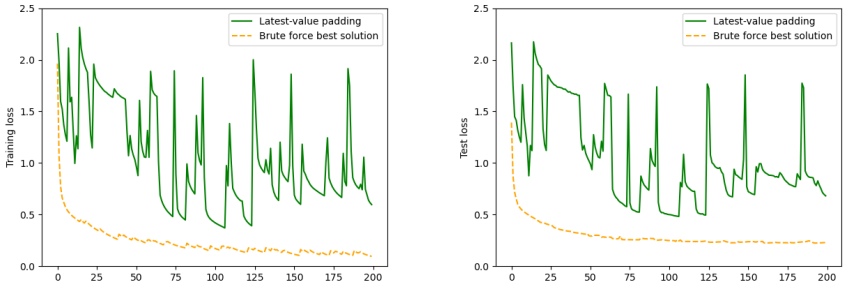


Fig. 12: Performance on latest-value padding.

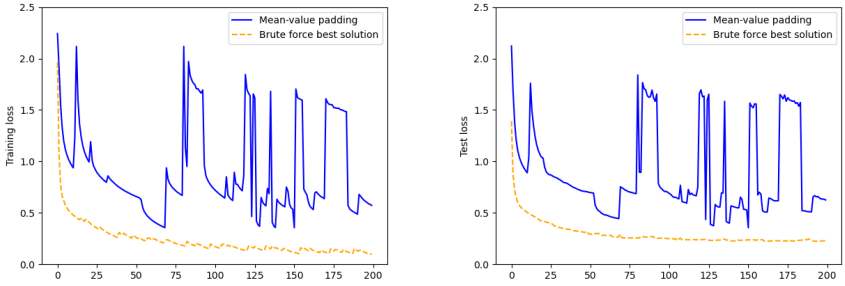


Fig. 13: Performance on mean-value padding.

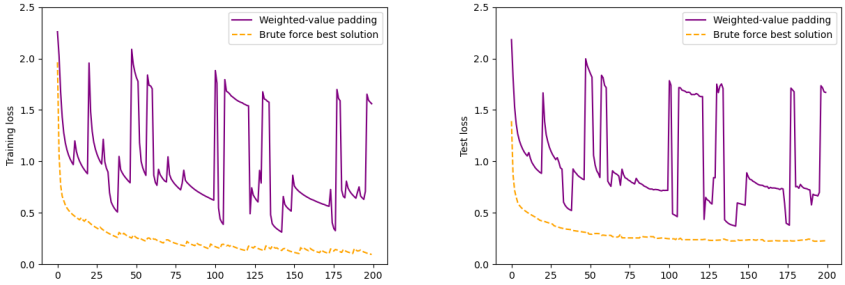
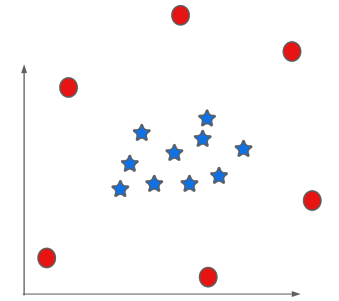
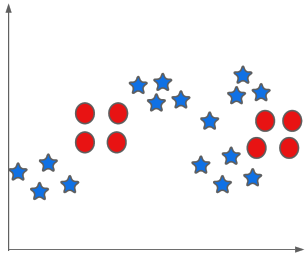


Fig. 14: Performance on weighted moving average padding.

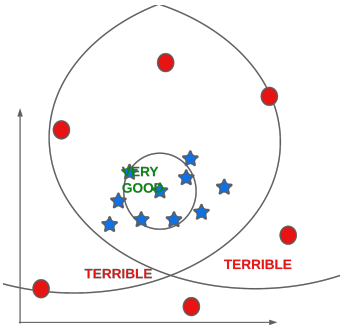
D Parzen MI Estimation examples



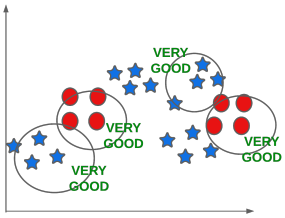
(a) Example 1



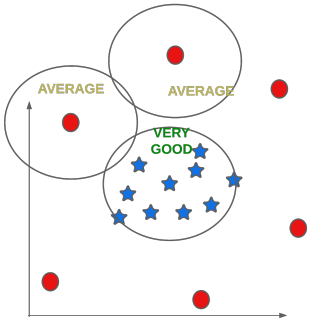
(b) Example 2



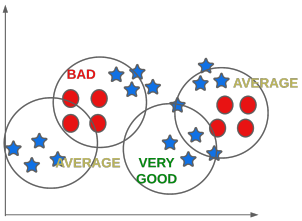
(c) KNN Example 1



(d) KNN Example 2



(e) Parzen Example 1



(f) Parzen Example 2

Fig. 15: Comparison of KNN (for $K = 3$) and Parzen behavior in two mock examples. The Parzen estimator would prefer example 1, while the KNN one would give a higher score to the second one. Both examples have a similar amount of datapoints.

References

[1] Gheda, R.: Dynamic VF-PS. https://github.com/r-gheda/Dynamic-VFPS

[2] Jiang, J., Burkhalter, L., Fu, F., Ding, B., Du, B., Hithnawi, A., Li, B., Zhang, C.: Vf-ps: How to select important participants in vertical federated learning, efficiently and securely? In: Advances in Neural Information Processing Systems (2022)

[3] OpenReview VF-PS Web Page. https://openreview.net/forum?id=vNrSXIFJ9wz Accessed 2023-06-18

[4] Romanini, D., Hall, A.J., Papadopoulos, P., Titcombe, T., Ismail, A., Cebere, T., Sandmann, R., Roehm, R., Hoeh, M.A.: Pyvertical: A vertical federated learning framework for multi-headed splitnn. arXiv preprint arXiv:2104.00489 (2021)

[5] OpenMined: PyVertical. https://github.com/OpenMined/PyVertical

[6] Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on ot extension. ACM Transactions on Privacy and Security (TOPS) **21**(2), 1–35 (2018)

[7] Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on {OT} extension. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14), pp. 797–812 (2014)

[8] Ipsen, N.B., Mattei, P.-A., Frellsen, J.: How to deal with missing data in supervised deep learning? In: ICLR 2022-10th International Conference on Learning Representations (2022)

[9] Chai, X., Gu, H., Li, F., Duan, H., Hu, X., Lin, K.: Deep learning for irregularly and regularly missing data reconstruction. Scientific reports **10**(1), 3302 (2020)

[10] He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 16000–16009 (2022)

[11] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[12] Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)

[13] Walters-Williams, J., Li, Y.: Estimation of mutual information: A survey. In: Rough Sets and Knowledge Technology: 4th International Conference, RSKT 2009, Gold Coast, Australia, July 14-16, 2009. Proceedings 4, pp. 389–396 (2009). Springer

[14] Kwak, N., Choi, C.-H.: Input feature selection by mutual information based on parzen window. IEEE transactions on pattern analysis and machine intelligence **24**(12), 1667–1671 (2002)

[15] Chollet, F.: Deep Learning with Python, p. 101. Manning Publications Co., ??? (2018). Chap. 3.1.4