Comparison of Federated Learning Scenarios: Vantage6 and PyTorch Distributed for Privacy-Preserving Learning

submitted in partial fulfillment for the degree of master of science

Simon Tokloth
11764066

master information studies
data science
faculty of science
university of amsterdam

2021-07-21

	First Supervisor	Second Supervisor
Title, Name	MSc Saba Amiri	Dr Adam Belloum
Affiliation	UvA, MNS	UvA, MNS
Email	s.amiri@uva.nl	a.s.z.belloum@uva.nl

Universiteit van Amsterdam

Comparison of Federated Learning Scenarios: Vantage6 and PyTorch Distributed for Privacy-Preserving Learning

Advantages and Disadvantages of an Open-source Federated Learning Infrastructure

Simon Tokloth
University of Amsterdam
simon.tokloth@student.uva.nl

ABSTRACT

This study aims to evaluate the performance of federated learning framework vantage6 and compares it to a generic approach to distributed machine learning by using PyTorch Distributed which is a parallel training package. Federated learning poses a novel way to perform distributed learning in machine learning scenarios which negates the need to aggregate all data in one location and instead performs the computation where the data is already stored. Big data is increasingly used in machine learning, and it often leads to better model performance and higher accuracy. However, large amounts of data often contain sensitive information. This thesis focuses on implementing a deep learning scenario on novel federated learning infrastructure vantage6 which aims to preserve privacy. Furthermore, local differential privacy and model encryption aim to enable further privacy protection. Both torch.distributed and vantage6 employ the same dataset and PyTorch model in this study. This research concludes that vantage6 offers better performance in terms of processing time while achieving equal accuracy scores across both implementations. Vantage6 is still in development and caused various incompatibility issues that are outlined throughout this thesis.

KEYWORDS

Federated Learning, Computer Vision, Privacy-Preserving Learning, PyTorch, Vantage6

1 INTRODUCTION

1.1 Privacy Issues of Machine Learning and Big Data

Conventional centralised Machine Learning (ML) has privacy-related issues. To ensure accurate model predictions and gain valuable insights, some features of a dataset that a ML algorithm trains on can contain sensitive information. One prominent type of sensitive data is health-related data [1] which can contain personal medical records or individuals. Big data complicates the situation by making centralised machine learning require even more computational resources. The problem of big data is that datasets are becoming too large and complex to be processed by conventional data processing methods. As amounts of data increase, it becomes more obscure whether data is sensitive or not. Big data is difficult to store and train. Therefore, centralised approaches to ML are becoming increasingly outdated and require improved methods.

Machine learning is increasingly used in the medical sector. Its predictions can help medical professionals prevent diseases and conditions or diagnose them in early stages [6]. However, while hospital patients might benefit from historical patient data in diagnosing their condition, sensitive information is at risk of being shared unintentionally or for commercial goals. In 2016, the European Union introduced the "General Data Protection Regulation" (GDPR) [13] which made the proper use and storage of sensitive data mandatory. These compliance guidelines are extensive and strict regulations to ensure that sensitive data are used and stored correctly. One example of the insufficient protection of sensitive data is the case of the Dutch hospital OLVG which resulted in a fine[1].

1.2 Federated Learning as Potential Solution

Federated learning (FL) is one potential solution to fixing some of those issues. To explain why privacy-preserving learning is of high importance in an increasingly data-driven world, a high-level introduction to privacy issues in ML was given in the previous section. FL decentralises the training of algorithms and executes an algorithm at the site where the local data is stored. Privacy loss can be minimised with comparable performance to centralised learning. This decentralised learning method was introduced by Google [9] in 2017, which indicates a shift in privacy management by big tech companies that often contribute to privacy concerns [4]. The use of distributed data centres can optimise the training process across locations and therefore require less computational resources [11]. Organisations that train their data as part of a collaboration in parallel with other organisations only communicate an updated model to a coordinating server. As a side-effect, this decentralised training adheres to data privacy regulations like the data minimisation principle of the GDPR. The reason is that only the updated model that was retrieved from the organisations is shared with and processed at the central server. The locally trained models are not stored after being sent to the central server as a global model which adheres to the storage and purpose limitation principles of the GDPR [11, 13].

This form of collaborative and decentralised (federated) ML is crucial in the healthcare sector since health data is categorised as one of the most sensitive information available according to [12]. Furthermore, as health information of a patient both benefit the patients themselves, as well as future patients, anonymisation would be a detriment to those benefits as that historical data only benefits a patient if it can be traced back to them. It can benefit a patient in diagnosing their condition based on former medical conditions. Therefore, the protection of that locally stored data is the best way to keep that sensitive data secure. FL does not require sensitive data to be shared or stored in a central location or database.

[1] https://edpb.europa.eu/news/national-news/2021/dutch-dpa-fines-olvg-hospital-inadequate-protection-medical-records_en

Thus, FL is often associated with clinical data [12]. A more detailed explanation of FL can be found in section 3.

1.3 Vantage6: A Better Alternative?

One particular infrastructure that provides a framework that combines decentralised learning with privacy protection is the FL infrastructure vantage6. The developers are creating a FL infrastructure in collaboration with the "Netherlands Comprehensive Cancer Organization" (IKNL) which deals with sensitive data of cancer patients [10]. Vantage6's privacy-preserving platform enables researchers and developers to implement ML methods while automatically adding privacy-preservation methods. While the environment needs to be set up by the developer, the FL communication is handled by vantage6 itself.

Vantage6 is designed with modular programming. Various different modules contribute to the entire framework. The infrastructure is open-source and employs Python coding and packages. A vantage6 workflow or environment consists of various researchers. One researcher is required to send one or multiple tasks, as a Docker image, to a central location from where the tasks are distributed to participating nodes. Docker is a tool that lets the researcher build an image and sends it to the nodes. The tasks are wrapped in one master task within the Docker image. One coordinating researcher ensures which nodes are allowed to participate in a FL scenario. Vantage6 provides these methods via RESTful API [10]. The central location is then responsible for the communication between the participating nodes by sending the tasks, retrieving results, and potentially adjusting those results at the central location. During those interactions, no sensitive data is allowed to be shared between the server and the nodes. All nodes perform the retrieved algorithm on their own locally stored data. Figure 1 indicates one database next to every participating node. The client hosts the central server and performs the communication.

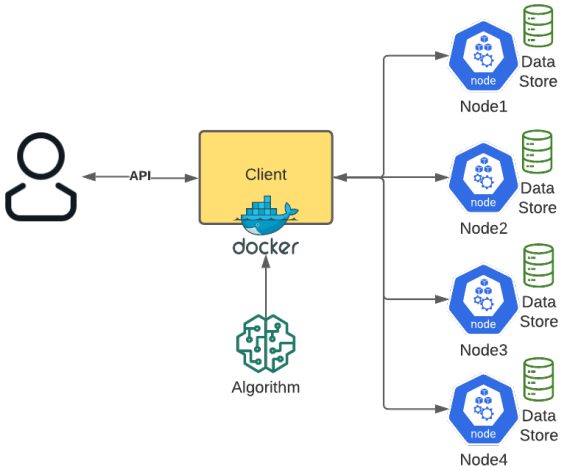

Figure 1: Basic vantage6 infrastructure.

The main purpose of this study is to examine how well vantage6 performs in comparison to an existing FL implementation using PyTorch Distributed (torch.distributed). While vantage6 offers automatic backend coordination and communication, torch.distributed requires developers to implement more communication code themselves. In vantage6, a developer only needs to specify what should be sent to and returned from a node. Both implementations will be evaluated according to various performance measures which are detailed in section 4. Section 4 also outlines the implementation and experimental setup of this study and how those differ between the two infrastructures.

Section 3 introduces existing literature in the field of FL. Furthermore, it elaborates on the infrastructures. Section 5 explores the results of the experiments according to the measurements introduced in the methods. The last three sections 6, 7, and 8 conclude this paper and offer insights and suggestions for future work while reflecting on findings. The research objectives that will employ these methods are introduced in section 2.

2 RESEARCH QUESTION

To compare the two infrastructures vantage6 and torch.distributed, this thesis requires the implementation of a FL scenario to be used on the same dataset. The following research question was formulated:

Does federated learning infrastructure vantage6 provide an advantage over torch.distributed in a PyTorch deep learning scenario?

The following sub-questions aim to support and deepen the understanding of the main research question as well as providing more detailed insights into the difference of the infrastructures:

(1) What are the advantages and drawbacks of vantage6 in terms of:
   - usability
   - documentation
   - implementation
(2) Does model performance suffer from transforming the dataset into CSV files?
(3) What are the bottlenecks in the current version of vantage6?
(4) Which infrastructure offers better privacy protection and encryption?
(5) Does local differential privacy (DP) cause a major detriment to model performance?

3 RELATED WORK

3.1 Federated Learning

Unlike centralised learning, FL does not use the collected data at a single data centre. Instead, it trains a model with local data over a distributed network of mobile devices and learns a shared model by adding local updates together. As the learning system is decentralised, Google coined this approach "federated learning" [9]. Although introduced for mobile devices with the intent to train algorithms on local data on smartphones, the same framework can be applied to institutions that have data stored locally.

The paper "Deep Learning with Differential Privacy" [1] describes a similar architecture with differential privacy (DP) and a decentralised method that communicates between different loosely federated devices that are coordinated by a central server. The

individual datasets are stored on those various nodes and never reach the central server, but are locally processed at those nodes which return results to the central server. What makes this privacy-preserving is the fact that due to the various updates to the central server's global model, the data on the local devices is deleted after the update to the server. Hence, FL reduces privacy loss by limiting the possibility of a security breach to only one of the many devices or nodes [1].

One major concern that the authors raised in [12] is that FL settings require the communication of model parameters, weights, and updates between the participating nodes. To evaluate that issue in FL, the authors compare it to the conventional centralised approach with clinical predictions. They find that stochastic gradient descent (SGD) with DP (DP-SGD) is easily applicable in centralised approaches, while still offering high privacy protection. Contrarily, they argue that the same implementation is more problematic in FL techniques.

However, other works reject that statement [1, 2], arguing that a solution to introduce privacy to ML is the use of DP-SGD. It is a common method used in ML to guarantee privacy and it can work in addition to FL [1]. The authors argue that composability, group privacy, and robustness to additional data make DP useful in privacy-preserving settings. During the optimising SGD steps, which is also employed as the optimizer of the PyTorch model in this study, noise is added for privacy protection. In PyTorch, SGD can easily have local DP attached to it. `Opacus`[2] is a Python library that helps to train ML models with DP. According to [2], FL alone does not protect the privacy of patients entirely as collaborating nodes often need to share intermediate results which can potentially contain patient data. Despite neither torch.distributed nor vantage6 allow for the sharing of those data, local DP ensures extra privacy-preservation.

Another paper elaborately states the advantages and unsolved issues of FL [5] which constitutes the research motivation of this thesis. As the need for privacy-preserving learning is growing, frameworks that enhance privacy need evaluation and testing. This can be achieved by analysing new frameworks that focus on FL. The authors indicate that focused data collection and data minimisation, namely only collecting as much data as needed, are a main focus of FL. Data minimisation can already result in solving a main principle of the GDPR guideline. One significant drawback that the authors mention is that the server poses the only major risk of privacy loss as one server is still required to coordinate the different devices. If the central server fails, the entire system collapses [5]. On the other hand, if one node fails, the entire system will not be greatly impacted by that.

3.1.1 Federated Averaging. Federated averaging (FedAvg) is a method specifically used in FL environments. In FedAvg, nodes return the trained parameters which are then averaged at the central location before they are again sent to the nodes to be trained again with averaged parameters. The central server aggregates the changes, for instance, parameters like weights, received from all the nodes. Then, the central server averages the parameters with the weights of the model. The devices train the model using the gradient descent algorithm, and the trained weights are sent back to the server [9]. FedAvg is one of the most used and fundamental methods in the FL framework [3]. Furthermore, FedAvg has been proven to be robust, as well as accurate of FedAvg [14].

[2]Retrieved from: https://opacus.ai

3.2 PyTorch Distributed

The baseline infrastructure of this study makes use of the PyTorch packages `torch.distributed` and `torch.multiprocessing`. While vantage6 is still in development and not distributed to many developers yet, torch.distributed has been examined and used often which makes it the ideal infrastructure to compare vantage6 to. Therefore torch.distributed can be used as a benchmark for FL as it was used to create the baseline FL pipeline. Torch.distributed uses data-parallel training which is a sub-package of PyTorch itself and thus expected to work ideally with PyTorch implementations. The intention of deep learning settings is that large amounts of data, as well as complex models, are valuable to good model performance [7]. Therefore, data-parallel training is useful when those large amounts of data and complex models are available. Since in torch.distributed each node trains on their own data, the computational resources are distributed among many participating nodes while still running the same algorithm and accumulating varying results to a global model.

3.3 Vantage6

The core of this paper is the exploration of how the FL infrastructure vantage6 compares to the implementation with torch.distributed using local DP package Opacus and the FedAvg algorithm. Vantage6 is an infrastructure that has not been researched and explored as much as torch.distributed or other decentralised training frameworks. The developers of vantage6 provide an academic paper alongside the documentation [10]. The documenting paper provided on the vantage6 website mainly deals with FL in the field of cancer informatics. As medical data is of sensitive nature, the privacy-preserving FL infrastructure is a framework applicable to this type of data. The FL architecture in vantage6 can be best described as flexible, user-friendly, and robust [8, 10]. This is closely related to the sub-question that aims to evaluate the ease of use or application of FL infrastructures. Vantage6 is open-source and is specifically addressed to cancer epidemiologists. One important aspect of using data that includes sensitive information on patients is the use of learning algorithms in health care to optimise health care provided to a patient. This can be done while preserving the patient's privacy. The authors draw upon the above-mentioned GDPR and claim that many issues like centralised data storing and processing do not comply with the GDPR nor the California Consumer Privacy Act [10]. How vantage6 works is described in sub-section 4.6.

4 METHODS

The programming language Python is used for both torch.distributed and vantage6. Vantage6 also requires the RESTful application programming interface (API) for setting rules and permissions to users. The two implementations will mainly be compared with model performance and computation time. The data that will be used to train and test the models are the CIFAR-10 and MNIST datasets. Both

MNIST and CIFAR-10 datasets are commonly used in computer vision projects. The given PyTorch model needs to be implemented as similar as possible in torch.distributed and vantage6. This ensures similarity and fairness for the comparison. The PyTorch model is a simple convolutional neural network (CNN) that employs SGD as the optimiser and local DP for privacy protection. In summary, the methodology will look as follows:

(1) Familiarise with and both introduced datasets according to [1, 9]
(2) Evaluate torch.distributed baseline infrastructure with PyTorch model
(3) Based on baseline infrastructure, implement PyTorch model in vantage6 infrastructure with DP-SGD
(4) Adjust baseline infrastructure according to limitations in vantage6
(5) find the bottleneck(s) of each infrastructure according to processing time, computational resource (GPU/CPU) usage, and model performance (test accuracy)
(6) Report on findings and main differences

Four experiments will be conducted to find potential bottlenecks. The different experiments are as follows:

- torch.distributed vs vantage6 using GPU
- torch.distributed vs vantage6 vs dockerised vantage6 using CPU
- difference MNIST and CIFAR-10 in torch.distributed vs vantage6
- performance issues with local DP in torch.distributed vs vantage6

Here, *dockerised vantage6* refers to using the actual client as opposed to the mock client. The mock client simulatees the presence of a server and node. As no virtual machines (VM) are employed for the experiments, both vantage6 and torch.distributed are mocking situations in which vantage6 uses a mock client and torch.distributed uses the torch.multiprocessing method `.spawn` to simulate multiple nodes. This allows for an accurate comparison of both infrastructures without using VMs. The dockerised vantage6 implementation makes use of the real client and Docker to distribute the Docker image to the node.

4.1 vantage6, torch.distributed, & dockerised vantage6

The reason that this study employs experiments with both vantage6 and dockerised vantage6 is because of compatibility and dependency issues with the vantage6 client package. Those issues that are present in the actual client work differently in the mock client that. In the mock client, vantage6 simulates the presence of nodes like torch.distributed does with `.spawn`. To still compare GPU usage and processing time as well as local DP, the vantagae6 mock client implementation will be compared to torch.distributed.

4.2 Metrics

The metrics *processing time*, *computational resource (GPU/CPU) usage*, and *model performance* aim to show the differences of the infrastructures. Model performance, namely test accuracy, will bring the smallest insight into the different performances of the infrastructures as both infrastructures use the same PyTorch model. Therefore, model accuracy should be nearly identical. The processing times, namely how much each implementation takes to iterate over the same amount of epochs, as well as how much of the GPU and CPU it uses are the important metrics of this study. They indicate which implementation runs more efficiently. Python package `time` was used to measure the computation time of the entire training including the initialisation. The package `psutil` was employed to measure virtual memory as well as CPU usage. Finally, `nvidia-smi` measured the GPU as well as VRAM usage.

4.3 MNIST & CIFAR-10

The MNIST[3] dataset is the primary dataset of the experimentation as it is the computationally less demanding dataset due to the lack of the RGB dimension. It contains grey-scale images of handwritten digits (between 0 and 9) and has a training set of 60,000 images and a test set of 10,000 images. The digits have been size-normalized and centred in a 28x28 image.

The CIFAR-10[4] dataset consists of 60000 32x32x3 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The CIFAR-100 dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). The 10 fine classes are considered the sensitive attributes. In CIFAR-100 the coarse classes are used for the target task and the fine classes as sensitive attributes.

4.4 Data Preprocessing & Distribution

The original intent for the datasets was to provide the model with independent and identically distributed random variables (fully-i.i.d) data distribution. While functioning in torch.distributed, PyTorch files with the ending .pt or .pth are currently incompatible with vantage6. The only readable file format is the "comma-separated values" (CSV) format in vantage6 which limits the available data distribution options. Therefore, the only data preprocessing applied to the MNIST dataset were the merging of the train and test set as well as the application of random shuffling. As PyTorch usually works with .pt files, a different method for the data preprocessing needed to be used which is not required for .pt files. The shuffled datasets needed to be loaded as .pt files and then reshaped. Then, the data needed to be converted to a NumPy array. The targets or labels of the dataset needed to be transformed to a list and concatenated with the dataset that was converted to a Pandas dataframe before it was saved as a CSV file.

Figure 2 shows the training set as a dataframe. The final employed MNIST dataset has a size of (70000 x 785) due to the merging of the training (60000 x 785) and testing (10000 x 785) sets. This results in 784 pixels for each row with a label attached to it. The first value is the label (a number from 0 to 9) and the remaining 784

[3]Retrieved from: http://yann.lecun.com/exdb/mnist/
[4]Retrieved from: https://www.cs.toronto.edu/ kriz/cifar.html

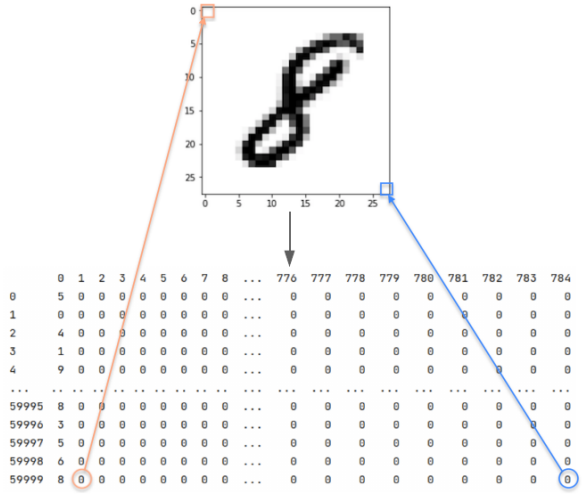

Figure 2: MNIST train set .pt to .csv transformation which results in a dataframe with the size (60000 x 785)

values are the pixel values (a number from 0 to 255). The preprocessing of the CIFAR-10 dataset differed from the MNIST dataset as it already contains the targets as a list and the data a NumPy array. Before training the model, both datasets needed to be transformed back to their original state, namely (70000 x 28 x 28) and (60000 x 32 x 32 x 3) for MNIST and CIFAR-10 respectively.

4.5 Setup

Torch.distributed employs a single-program architecture and vantage6 requires multiple programs which results in various differences. One local machine is used for all experiments. As vantage6 uses multiple sub-tasks, computation becomes demanding. Therefore a remote server GPU was used in order to ensure sufficient CUDA memory. However, that remote server was not used during the vantage6 implementation through Docker. As the vantage6 experimentation with multiple nodes will be executed locally, with a local Docker image, a remote server cannot be employed.

4.6 Vantage6 Infrastructure

Vantage6 is still in development. Therefore, the infrastructure lacks many functions that torch.distributed provides. The functionality of all components, such as the command-line interface (CLI) or the modules, is not always guaranteed in vantage6. The most essential one that is lacking is the availability of node-to-node communication. Node-to-node communication is the updating of parameters and communication between nodes. It is not necessarily required in federated learning. However, when employing the federated averaging method it is necessary for the updating of the parameters between the nodes. In vantage6, multiple sub-tasks needed to be used to perform federated averaging iteratively as a workaround. The infrastructure then looks as follows:

Figure 3 displays the implementation of the PyTorch model in the vantage6 infrastructure. In vantage6, a central server coordinates the communication between the user(s) and nodes. It does this by exposing a RESTful API that allows the main researcher to create tasks. Those tasks, encapsulated in a master task, are computation requests that are sent to the nodes. Those nodes can be hospitals or other organisations that have data that they want to put through the main algorithm without sharing the data with the central server. These individual sub-tasks are executed one after the other by each participating organisation (node). Then, the results are returned to the server. Those functions that are executed at the nodes are coloured blue in figure 3 and those functions and methods that happen at the central location are coloured red. The server can support multiple organizations, collaborations and users. An organisation can therefore have multiple users and organisations make up one collaboration. Those nodes need to provide the required processing capacity as the computation takes place at each node individually. In the algorithm that the node receives, the user is required to insert their own dataset as part of the node configuration file. A node cannot alter the hyper-parameters, which ensures that all nodes train on the same PyTorch model. Further methods can alter those returned results and average them in order to improve a model by averaging the parameters returned by all nodes by using the FedAvg algorithm.

The nodes receive the algorithms through a Docker image. That image is built using a Docker file that needs to be located in the project's package structure. That image is uploaded to a Docker hub or registry through which it can then be accessed by the remote nodes. A Docker image can be considered a snapshot of the algorithm. As soon as a node receives a task from Docker, that node downloads the docker image which contains the algorithm with its master task and sub-tasks and then runs the algorithm. Afterwards, the results are returned through Docker to the server. End-to-end encryption for all messages is possible for extra data security in vantage6.

By adding the prefix "RPC_" to a function, the vantage6 backend communication automatically regards a function to be a function that needs to be executed at the nodes. Functions without that prefix will not be executed at the nodes. Each RPC_method requires the first argument to be "data" and it only accepts one data argument. Therefore, it also only accepts one dataset from each node configuration.

Due to the nature of vantage6, an identical copy of PyTorch's torch.distributed could not be implemented. This lead to key differences. In vantage6, each execution is a sub-task of a master task. While in torch.distributed it is easy to update the parameters with torch.distributed native functions, node-to-node communication is not possible in vantage6. Therefore, each communication step needs to go through the client. The functions that vantage6 does not offer yet are to be added in version 3.0.0. The following algorithm shows the adjusted FedAvg method for vantage6. It has the same functionality as the function in torch.distributed, though requires another sub-task:

Algorithm 1 also returns the model as well as the test accuracy as a dictionary. That is due to vantage6's architecture. As training and testing is one task and executes either depending on the given argument, the function needs to return the model for the next testing execution. During the execution, vantage6 cannot store files to the disk nor access them. Therefore, it takes that model

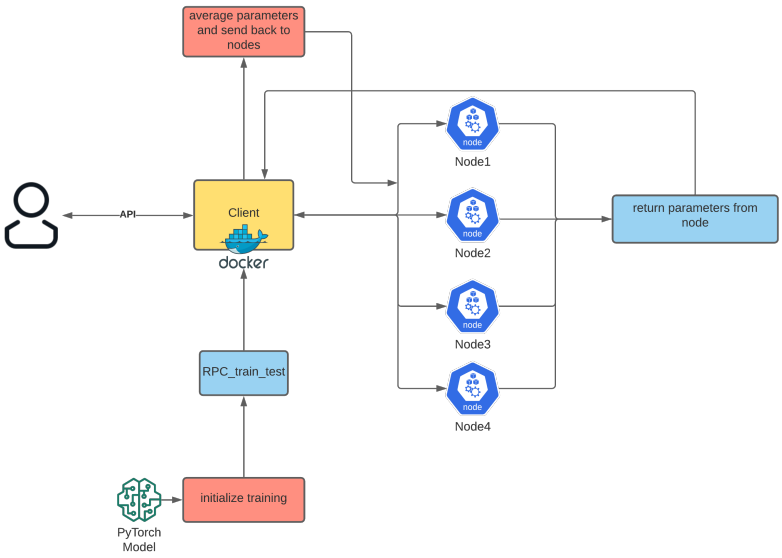

Figure 3: The infrastructure visualised as a diagram.

Algorithm 1 Federated Averaging in vantage6

```
for parameters in list(model.parameters()):
return 'params': parameters,
'model': model,
'test_accuracy': test_accuracy
result = client.get_results(task)
for output in result:
global_sum += output["parameters"]
global_count += len(global_sum)
averaged_parameters = global_sum / global_count
```

as a dictionary in the next sub-task. The test accuracy needs to be returned as a dictionary so it is sent to the server as the result. The Docker container that shows the testing would still show the test accuracy, but the client would not return it as the result of the node's testing.

5 RESULTS

5.1 Processing times using GPU

The first experiment includes the mock client version of vantage6. It is the most similar implementation to torch.distributed. Additionally, it is the only environment that allows for the use of CUDA. CUDA is a parallel computing platform and application programming interface model developed by Nvidia. Docker does not recognise local NVIDIA drivers which was not possible to bypass in the experiments. Due to its iterative sub-task nature, vantage6 requires far more memory of the graphics card (VRAM) in vantage6 than in torch.distributed. Each sub-task requires less VRAM than torch.distributed, however, torch.distributed is one task by nature, whereas vantage6 requires at least two sub-tasks for FedAvg, while torch.distributed just updates (`.gather()` and `.broadcast()`) the parameters of the nodes.

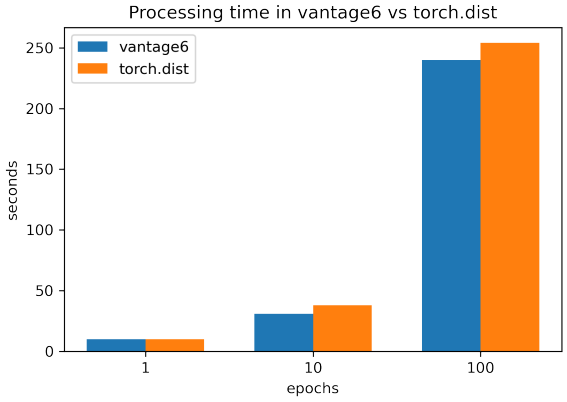

Figure 4: Processing time in vantage6 in comparison to torch.distributed running on GPU without differential privacy

Figure 4 shows the results of 1, 10, and 100 epochs in vantage6 and torch.distributed (torch.dist). Evidently, both infrastructures have similar processing times. A reason why vantage6 runs a little faster than torch.distributed might be due to the fact that it uses no extra Python file for parsing as the arguments are specified in JSON format in the sub-tasks themselves. As communication between the server and the nodes happens in the backend, vantage6 seems to communicate more efficiently.

Figure 5 shows the same experiment setting, now with the Opacus privacy engine (local DP) attached to the optimiser. It leads to the same conclusions, namely that the processing time differences

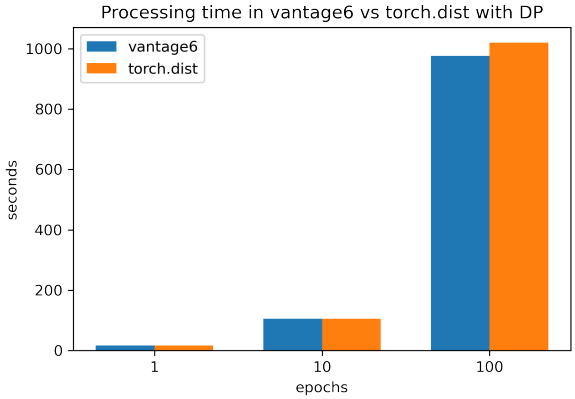

Figure 5: Processing time in vantage6 in comparison to torch.distributed running on GPU with Opacus Privacy Engine

Table 1: MNIST GPU with local_dp in vantage6.

	Accuracy	Epsilon	Alpha	Seconds
1	68	0.551	23	17
10	88	0.639	22	105
100	92	1.4133	18	976

between the two environments are minor. With local DP, the differences are even smaller. The exact times, as well as accuracy results of the experiments, are shown in tables 1-4.

Table 2: MNIST GPU with local_dp in torch.distributed.

	Accuracy	Epsilon	Alpha	Seconds
1	58	0.551	23	16
10	85	0.639	22	105
100	92	1.4133	18	1020

Table 1 contains the detailed results when using the remote GPU in vantage6 as opposed to table 2 that shows the same results for torch.distributed. Evidently, while fluctuations occurred, the computation times are nearly identical as evaluated above. However, while using the same amount of privacy budget epsilon and getting the same best alpha, the vantage6 implementation shows improved accuracy over the torch.distributed implementation. Epsilon quantifies the privacy properties of the algorithm. Epsilon, delta and alpha values return a quantification of the privacy guarantee of DP and the privacy budget the algorithm has spent. The parameter alpha tells the Opacus privacy engine which DP order to use to track the privacy spending. The local DP seems to have a higher impact on the torch.distributed implementation. However, that difference becomes less prevalent the more epochs are used. For 100 epochs, both implementations perform equally well.

Table 3: MNIST GPU without local_dp in vantage6 (left) & torch.distributed (right).

	Accuracy	Seconds
1	93	10
10	98	31
100	99	240

	Accuracy	Seconds
1	91	10
10	97	38
100	99	254

The tables in table 3 indicate that accuracy and computation time are similar with vantage6 having a slight advantage over torch.distributed. This indicates that vantage6 is more efficient. This part answers the sub-question 5, which asked whether local DP causes a major detriment to the model performance. Execution times are considerably higher in the experiment with local DP, and the test accuracy is considerably lower due to the added noise. With a high amount of epochs, both vantage6 and torch.distributed reach test accuracy of more than 90%.

5.2 Federated Averaging GPU usage

As the federated averaging implementation was different for vantage6, some issues occurred with the use of CUDA. The GPU usage was approximately 60% for both implementations. While vantage6 used 2.2 GB of VRAM for the first sub-task, torch.distributed used 3.7 GB in total. However, when iterating over the second sub-task which trains with the averaged parameters in vantage6, the program used 7.2 GB of VRAM before throwing an error that CUDA ran out of memory. This is an issue as CUDA does not clean up its allocated memory which is necessary for vantage6's iterative updating of parameters. Federated averaging did not affect the accuracy of the training in either implementation. This part offered an insight into the bottleneck of vantage6 as asked in sub-question 3. As processing times showed, vantage6's architecture does not appear to have any processing time bottlenecks. Various compatibility issues outlined in sub-section 5.4 cause limitations of the infrastructure, however, the workarounds do not cause any performance issues except for exhausting hardware resources.

5.3 Processing times using CPU

This section includes the addition of the dockerised vantage6 implementation. All implementations used on average 50% of the CPU with nearly identical virtual memory usage between vantage6 and torch.distributed, while the dockerised vantage6 implementation uses the entire virtual memory available. Unlike CUDA, the CPU freed its cache and allocated virtual memory automatically after each process in vantage6. Therefore, the bottleneck that appeared with the GPU experiment is not applicable to CPU processing. Nonetheless, GPUs are preferred over CPUs in image classification settings.

The dockerised installation using the components is the preferred build of vantage6 and is also the final version of the vantage6 implementation in this study. The training performance measured in accuracy between all infrastructure settings was similar. However, as seen in figure 6, the dockerised vantage6 is considerably more efficient than its mock client version as well as the torch.distributed implementation. Evidently, dockerised vantage6 takes the longest

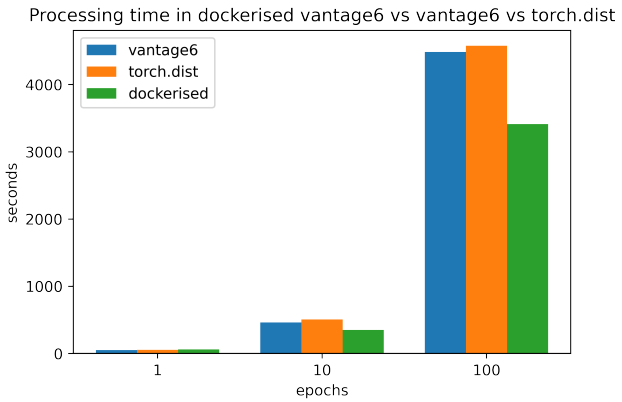

Figure 6: Processing time in dockerised vantage6 in comparison to vantage6 and torch.distributed running on CPU

to initialise, as seen in the slightly higher runtime with epoch = 1 in comparison to the vantage6 with the mock client as well as torch.distributed. Afterwards, the dockerised implementation runs more efficiently and faster. This shows that vantage6 offers a significant performance improvement. Lower runtimes enhance usability. With further development of vantage6 and improved documentation, vantage6 could be a better alternative to PyTorch's native torch.distributed package.

Table 4: Computation speed across all three implementations.

	1	10	100
vantage6	51	461	4484
torch.dist	54	507	4578
dockerised v6	59	349	3414

As the dockerised version hosts both the master container as well as the node container, the virtual memory is exhausted more than in mock client vantage6 and torch.distributed. Virtual memory is the virtual process that the system synthesises to represent the memory and CPU resources consumed by the programme. While the advantage of the mock client implementation performed slightly better with the GPU than torch.distributed, this part showed a major improvement over the torch.distributed implementation. This answers the main question which asked whether vantage6 offers an advantage in this PyTorch implementation.

5.4 Usability, Documentation, & Implementation

Vantage6 has many issues that need to be worked around. One main issue that was encountered during the implementation was due to the privacy engine of Opacus. As vantage6 works iteratively with sub-tasks, Opacus tries to attach the privacy engine to the model twice. That is not possible. The function `privacy_engine.detach()` does not provide a solution and seems to be malfunctioning in release 0.13.0. With the remote server, which has the same package versions installed, this error does not occur. In the actual implementation, local DP could not be used. When using the actual client, there is a dependency issue with `vantage6.client` and Opacus, namely that vantage6 requires the package `requests` to be exactly to be 2.23.0 and Opacus requires at least 2.25.1. Using previous releases of Opacus did not solve that issue. In the mock setting (with the remote server), local DP can be attached to both the first training iteration as well as the averaged training iteration.

Another issue occurred with the transformation of the CIFAR-10 dataset. While MNIST has one fewer dimension than CIFAR-10, as CIFAR-10 has an additional RGB layer, converting CIFAR-10 into a single CSV file is a considerable downside to the functionality of the PyTorch model. Therefore, the test accuracy was always 9.73% and did not train properly which indicates an error in the transformation or the requirement of an additional file which is not possible in the current version of vantage6.

In its current stage, vantage6 does not allow for direct data distribution in deep learning settings as it expects local data at each node which is how a real-case scenario would look like. Therefore, the fully-IID data distribution would need to be done beforehand manually to the CSV file and set to each VM. That was beyond the scope of this study. This part answers sub-question 2 which asked whether transforming the datasets into CSV files would affect the model performance negatively. While there were no differences between the .pt and CSV file in terms of model performance, CIFAR-10 could not be trained on due to the transformation of the dataset.

One incompatibility issue had to do with PyTorch's `model.parameters()` function. That method is a generator which needs to be used for the SGD optimisation of the model. Generators cannot be pickled by the Python `pickle` module employed in vantage6. However, converting those generators to a list solved the issue. Although an easy solution existed, it is a flaw in the vantage6 infrastructure that should be considered.

Lastly, an error that developed early on in the development phase had to do with averaging as part of sub-tasks in vantage6. As soon as the parameters of the model are adjusted, they are no longer leaf tensors. This means that they are no longer at the beginning of the graph. This is an issue in this context as the returned parameters from the training round at the nodes returns leaf tensors. Yet, as soon as those parameters are added up from all nodes and divided by the number of nodes that trained the model, the tensors are no longer leaf tensors. This causes an error in vantage6, as Opacus needs to be set up anew in the next training sub-task due to vantage6's architecture. The privacy engine of Opacus can only be attached to leaf nodes. To bypass an error that the optimizer would raise, the methods `.detach()` and `.clone()` need to be applied in order to make the optimizer accept the parameters as leaf tensors. This is an issue only present in the vantage6 implementation as it is caused by the FedAvg workaround.

Sub-question 1 aimed to identify which infrastructure is more usable. The issue with Opacus, as well as the issues addressed with the PyTorch sub-packages and data formats, hinder a developer to implement various third-party Python packages. Most issues were avoided with simple workarounds, however, some packages were impossible to use due to vantage6's architecture.

6 DISCUSSION

Although stated in [12] that applying DP to federated settings is more difficult than to centralised settings, modern tools like the Python package Opacus allow for easy integration of local DP in PyTorch, no matter the setting. Although the model performance is lower, and processing power suffers from added noise, an increased amount of epochs still results in high model accuracy as seen in section 5. Furthermore, this study displayed that high accuracy can still be obtained in FL settings while not sharing sensitive data with the central server. The more organisations participate, the more data is available which tends to result in more accurate predictions.

Both vantage6 and torch.distributed offer various advantages in the FL pipeline. Torch.distributed gives the developer more freedom and flexibility in setting up their own communication as well as not restricting the developer to a specific data type, whereas vantage6 simplifies the communication backend and automates the deployment of FL in real-case scenarios. Vantage6 without FedAvg appears to be faster than torch.distributed. However, when it comes to FedAvg, torch.distributed has the upper hand in the non-dockerised experiment. The reason for that is that due to vantage6's current architecture, no node-to-node communication is possible the same way as in torch.distributed. That is one of vantage6's major disadvantages. A similarly working node-to-node communication implementation is planned for release 3.0.0 of vantage6. Right now, vantage6 cannot update the parameters automatically between the nodes during a task (as each round is one sub-task), which makes it an intermediate process between two sub-tasks.

The possibility of easily adding extra encryption in vantage6 offers further privacy-protecting measurements. Therefore, it answers sub-question 4 that asked which infrastructure offers the better protection. Despite the data never leaving the local nodes, supplementary encryption of the image and node enables privacy protection. Thus, by default, vantage6 offers better privacy protection.

Throughout this study, vantage6 has shown various issues that were not apparent in the torch.distributed infrastructure. However, despite torch.distributed being designed to work well with PyTorch, vantage6 has great potential for FL in ML settings. One benefit is the argument parsing. In vantage6, the arguments and hyperparamereters are easy to change. However, whenever researchers want to change the parameters, they must build a new Docker image which can be time-consuming. Therefore, it is also a disadvantage and results in long waiting times between testing the algorithm as building a Docker image requires time. One advantage is the "RPC_" prefix method which tells the vantage6 backend that it is to be executed at the nodes only which simplifies server-to-node communication. For the further implementation of ML, especially deep learning models, it is advisable for researchers to wait for the 3.0.0 release of vantage6.

As a data science student, I am familiar with the concept of data privacy and deep learning library PyTorch. Throughout this study, I became more familiar with various file formats, API, and various computer science concepts. Since I have only learned privacy law and ethical issues about ML and privacy before this study, this thesis has been a great way to learn about actual implementations that tackle those issues. FL will be a major advantage in the field of ML because of the increasing awareness of data privacy and the surge of big data which will require FL for processing those large amounts of data. New infrastructures like vantage6 offer user-friendly options of establishing FL settings.

6.1 General Limitations

A major limitation of vantage6 is the restriction to one CSV file which makes the implementation of, especially deep learning settings challenging. Another issue of FL that was mentioned in the literature review section was that FL is highly reliant on the central server. A server failure of the central server would cause the entire system to fail. Since node-to-node communication is not available in vantage6, this might cause problems. Therefore, this is an open issue that vantage6 does not tackle yet since vantage6 relies on Docker and server-to-node communication. Since the use of iterative sub-tasks, rather than offering methods for updating parameters, high memory usage can cause the central server to delay result output. Lastly, as vantage6 is still in its development phase, few ML settings have been implemented and tested.

6.2 Project-specific Limitations

As PyTorch employs a specific format to store models and datasets, vantage6 did not support that native data format. Furthermore, during an iteration in vantage6, nothing can be written to the disk, therefore, a trained model cannot be stored and loaded for the testing during a process. These two obstacles made it impossible to make the train and test functions separate which resulted in RPC_train_test with an argument "if_test" for whether a sub-task is supposed to execute training of testing.

A limitation of this study is the absence of actual nodes simulated by VMs. This would lead to more accurate results by comparing a FL simulation with actual nodes that process their own data or parts of a data distribution with their own computational resources. Due to the absence of results of a secondary dataset, this study cannot fully confirm that vantage6 performs better than torch.distributed in image recognition solely based on the MNIST dataset. Further reserach is needed to affirm that hypothesis with additional computer vision datasets.

7 CONCLUSION

This study incorporated the implementation of PyTorch in vantage6. PyTorch has not yet been implemented in vantage6 by any other developer. This signifies the importance of this study for future research and contribution to the field of study.

Comparing the two infrastructures vantage6 and torch.distributed lead to many insights on the advantages and disadvantages of them in FL settings. While vantage6 performed better on a GPU and was faster than torch.distributed, the VRAM was heavily burdened by vantage6 framework. However, the CPU was not impacted by this. To conclude, the dockerised version of vantage6 is the fastest and arguably most secure infrastructure as the experiments showed and offers various benefits like encryption.

To answer the main research question asking whether vantage6 offers advantages over torch.distributed, the results show that the dockerised installation of vantage6 has a high advantage

over torch.distributed in terms of computation times. Despite hosting the central part of the algorithm, the node that receives the algorithm, and requiring Docker, the computation time of vantage6 was the lowest. The third-party Docker integration added to that efficiency. Furthermore, while torch.distributed is mainly intended for the data-parallel training of PyTorch models, vantage6 offers FL for a variety of statistical methods as well as ML methods. A main bottleneck of torch.distributed in this experimental setup was the single-program implementation as opposed to the modular architecture of vantage6 that allowed the training process to utilise all computational resources which resulted in more efficient runtimes. FedAvg does not undermine the model performance in either infrastructure, though due to its iterative nature in the vantage6 implementation, causes CUDA to run out of memory.

8 FUTURE WORK

Future work with vantage6 is highly encouraged. However, if a future researcher aims to work with vantage6 on deep learning systems that include PyTorch, they are advised to wait for the 3.0.0 release version of vantage6 and discuss beforehand whether the above-mentioned limitations are fixed or whether the workaround employed in this study will still need to be used.

In future projects, VMs should be employed to simulate participating nodes, with one VM representing one node. At those nodes, different partitions of a .pt PyTorch dataset could be attached to simulate the participating nodes contain different data. Future work can built on this research project, while alterations should be made as soon as node-to-node communication will be available in vantage6.

The different FedAvg employed in the vantage6 experiments did not affect the resulting accuracy of the training. However, since no data distribution beyond merging and shuffling the datasets was applied in the experiment, an experiment will need to be run as soon as vantage6 allows for additional data formats, especially in .pt format when PyTorch models will be implemented.

While this study focused on the comparison of the FL infrastructure and PyTorch data-parallel training package, further studies will investigate vantage6. That study will investigate additional ML methods within the vantage6 environment and aims to integrate that into Brane. Brane is a module that utilises containerisation to encapsulate functionalities as portable building blocks to improve accessibility to computer science novices by bypassing the backend computing.

9 ACKNOWLEDGEMENT

Throughout this thesis, I have received a great deal of support and assistance. I would like to thank my supervisors, Saba Amiri and Dr. Adam Belloum for helping me throughout my dissertation. I would like to acknowledge my co-students from my research group for their collaboration as well as the vantage6 developer group.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 308–318. https://doi.org/10.1145/2976749.2978318

[2] David Froelicher, Juan R. Troncoso-Pastoriza, Jean Louis Raisaro, Michel A. Cuendet, Joao Sa Sousa, Jacques Fellay, and Jean-Pierre Hubaux. 2021. Truly Privacy-Preserving Federated Analytics for Precision Medicine with Multiparty Homomorphic Encryption. *bioRxiv* (2021). https://doi.org/10.1101/2021.02.24.432489 arXiv:https://www.biorxiv.org/content/early/2021/02/25/2021.02.24.432489.full.pdf

[3] Wei Huang, Tianrui Li, Dexian Wang, Shengdong Du, and Junbo Zhang. 2020. Fairness and Accuracy in Federated Learning. arXiv:2012.10069 [cs.LG]

[4] Olly Jackson. 2019. GDPR: Google fine could lead to corporate structure changes. *International Financial Law Review* (Jan 25 2019). https://www.proquest.com/scholarly-journals/gdpr-google-fine-could-lead-corporate-structure/docview/2185452904/se-2?accountid=14615 Name - Google Inc; Copyright - Copyright Euromoney Institutional Investor PLC Jan 25, 2019.

[5] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2019. Advances and Open Problems in Federated Learning. arXiv:1912.04977 [cs.LG]

[6] Igor Kononenko. 2001. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine* 23, 1 (2001), 89–109. https://doi.org/10.1016/S0933-3657(01)00077-X

[7] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. arXiv:2006.15704 [cs.DC]

[8] Frank Martin, Melle Sieswerda, Johan van Soest, Arturo Moncada-Torres, intGRen, and Codacy Badger. 2020. *IKNL/vantage6: 1.0.0a1.* https://doi.org/10.5281/zenodo.3750881

[9] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629 [cs.LG]

[10] Arturo Moncada-Torres, Frank Martin, Melle Sieswerda, Johan van Soest, and Gijs Geleijnse. 2020. VANTAGE6: an open source priVAcy preserviNg federaTed leArninG infrastructurE for Secure Insight eXchange. In *AMIA Annual Symposium Proceedings* (2020). 870–877.

[11] Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. 2018. A Performance Evaluation of Federated Learning Algorithms. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning* (Rennes, France) *(DIDL '18)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3286490.3286559

[12] Stephen R. Pfohl, Andrew M. Dai, and Katherine Heller. 2019. Federated and Differentially Private Learning for Electronic Health Records. arXiv:1911.05861 [cs.LG]

[13] European Union. 2016-05-04. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal L110* 59 (2016-05-04), 1–88.

[14] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated Learning with Non-IID Data. *CoRR* abs/1806.00582 (2018). arXiv:1806.00582 http://arxiv.org/abs/1806.00582