

HEREDITARY

HetERogeneous sEmantic Data integration for the guT-bRain interplaY

Deliverable 2.15

Communication protocols

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No GA 101137074. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.



**Funded by
the European Union**

EXECUTIVE SUMMARY

Deliverable 2.15 on “Communication protocols” reports on the progress within Task 2.6 regarding communication protocols for data- and model-centric federated methods. In this deliverable, we report on how the Flower federated learning framework communicates during federated learning experiments and specifically how this communication can be secured. Specifically, we cover the following methods to secure communication:

- Secure Aggregation
- Differential Privacy
- Trusted execution environments

Secure Aggregation and Differential Privacy have been validated and disseminated during Project Workshop 5, organised virtually on October 1, 2025.

Based on the code provided to participants in workshop 5, we have concluded that the decrease in model performance is negligible for both Secure Aggregation and Differential Privacy. The runtime remains constant when utilizing Differential Privacy, whereas Secure Aggregation adds about 13% runtime due to the additional communication required between the server and clients. In addition, the operational overhead of deploying either technique is minimal when utilizing the Flower federated learning framework, although Differential Privacy requires model and dataset-specific tuning of the hyperparameters. Finally, we have concluded that Trusted Execution Environments can provide additional security guarantees for the global model, but require significant technical setup on both the server and all clients, which is highly dependent on the available hardware, making it impractical to deploy compared to Secure Aggregation and Differential Privacy.

DOCUMENT INFORMATION

Deliverable ID	D2.15
Deliverable Title	Communication protocols
Work Package	WP2
Lead Partner	SURF
Due date	31.10.2025
Date of submission	24.10.2025
Type of deliverable	OTHER
Dissemination level	PU

AUTHORS

Name	Organisation
Douwe van der Wal (Author)	SURF
Damian Podareanu (Author)	SURF
Bryan Cardenas Guevara (Author)	SURF
Elisabetta Biasin (Internal Reviewer)	KUL
Anna Romanovych (Contributor)	UNIPD

REVISION HISTORY

Version	Date	Author	Document history/approvals
0.1	10.09.2025	Douwe van der Wal	Introduction, Secure Aggregation and Trusted Execution environment section
0.2	24.09.2025	Bryan Cardenas Guevara	Differential Privacy section
0.3	24.09.2025	Damian Podareanu	Minor corrections and comments to improve deliverable
0.4	29.09.2025	Douwe van der Wal & Bryan Cardenas Guevara	Process comments and cosmetic improvements.
0.5	07.10.2025	Elisabetta Biasin	Revision and suggestions
0.6	08.10.2025	Douwe van der Wal	Implemented suggestions
0.7	17.10.2025	Douwe van der Wal & Bryan Cardenas Guevara	Minor changes to the Differential Privacy section
1.0	23.10.2025	Anna Romanovych	Finalisation of the layout of the deliverable

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Contents

Table of Abbreviations	7
1 Introduction	8
2 Workshop 5: Secure Aggregation and Differential privacy	9
2.1 Secure Aggregation	9
2.1.1 Using Secure Aggregation in Flower	10
2.1.1.1 Performance impact of using Secure Aggregation in Flower	11
2.1.2 Trusted Execution Environments	11
2.2 Differential Privacy	12
2.2.1 Differential Privacy in Flower	14
2.2.2 Performance Impact of using Differential Privacy	15
3 Communication within Flower	18
3.1 Flower messaging API updates	18
3.2 gRPC in Flower	18
4 Conclusion	19
References	20
Annexes	22

List of Tables

Table 1 Impact of Secure Aggregation on model performance and runtime	11
Table 2 Impact of Differential Privacy on model performance and runtime	15

List of Figures

Figure 1 How to configure Secure Aggregation on the server side in Flower	10
Figure 2 How to configure Secure Aggregation on the client side in Flower	10
Figure 3 DP-SGD algorithm (Abadi2016)	13
Figure 4 How to configure Differential Privacy on the server side in Flower	14
Figure 5 How to configure Differential Privacy on the client side in Flower	15
Figure 6 Graph of training loss showing the performance impact of Differential Privacy	17

Table of Abbreviations

Abbreviation	Expanded form
AAU	Aalborg University
AI	Artificial Intelligence
ALS	Amyotrophic lateral sclerosis
API	Application Programming Interface
CLEF	Conference and Labs of the Evaluation Forum
CPU	Central Processing Unit
CNN	Convolutional Neural Networks
DP	Differential Privacy
DP-SGD	Differentially Private Stochastic Gradient Descent
EEG	Electroencephalogram
EU	European Union
FL	Federated Learning
Flower	A Federated Learning Framework
GPU	Graphics Processing Unit
gRPC	Cross-platform high-performance remote procedure call
HES-SO	University of Applied Sciences and Arts Western Switzerland
i.i.d.	Independent and Identically Distributed
KUL	KU Leuven
MGF	Moment-Generating Function
ML	Machine Learning
RAM	Random Access Memory
REST	Representational State Transfer
SecAgg	Secure Aggregation
SecAgg+	Secure Aggregation +
SURF	Dutch National Supercomputing Centre
TEE	Trusted Execution Environments
TLS	Transport Layer Security
UNIPD	University of Padua (Università degli Studi di Padova)
UNITO	University of Turin (Università degli Studi di Torino)
UCD	University of Colorado Denver
VUB	Vrije Universiteit Brussel

1 Introduction

One of the primary objectives of the HEREDITARY project is to develop a federated learning and analytics infrastructure that functions within secure computing environments, including supercomputers. This configuration allows for distributed machine learning across multiple clients while prioritising data privacy and maintaining model security. By keeping data local, the system avoids centralisation, which is particularly crucial when handling sensitive medical data.

This deliverable reports on the work carried out so far in Task 2.6 (Federated learning infrastructure) which focusses on Federated learning infrastructure, where this deliverable specifically dives deeper into the communication protocols that are used in federated learning and how these enable secure and performant federated learning. As the federated learning field is rapidly developing, this overview of currently available methods to improve the safety of federated learning can help the project build upon existing work.

In this deliverable we outline secure aggregation, which allows the server to aggregate model updates from all clients participating in the experiment without the possibility of the server analysing a model update from a specific client. This provides additional protection for potential sensitive and/or private data used in an experiment. In short, clients create a pairwise mask with other clients in the experiment, which is cancelled out when combining all model updates.

Furthermore, this deliverable explores the implementation of differential privacy, a technique that introduces controlled noise to client model updates, thereby minimizing the disclosure of sensitive information about individual client data. By integrating differential privacy, the federated learning infrastructure can further enhance its security and privacy guarantees, ensuring the protection of sensitive medical data.

Finally, we will also dive deeper into the communication protocols used by the Flower framework (Beutel2020), namely gRPC, and how this can be secured using TLS with keys and provide an update on recent developments within the Flower framework that makes it easier for users to structure communication when utilizing the Flower Messaging API.

2 Workshop 5: Secure Aggregation and Differential privacy

2.1 Secure Aggregation

In horizontal federated learning experiments, clients participating in the experiment frequently share model updates with the central server. These updates contain the global model that has been fine-tuned on the local data of the client. The localized update can reveal details about the data distribution of the client or can in some extreme cases even be used to restore training data.

Secure Aggregation (SecAgg) (Bell2020) is a method that allows the central server to construct the new global model, without being able to gather any information from the update received by the client. This is achieved as follows:

1. First, each client creates a public and private keypair and all public keys are shared with all clients through the server.
2. Using the key agreement, each pair of clients establishes a shared random seed (one per pair). Each client can then generate a pairwise random mask vector with respect to each other client.
3. Then, the update is masked like so, where x is the weight update, N is the group

$$\vec{x}_i + \vec{r}_i - \sum_{j \in N_G(i), j < i} \vec{m}_{i,j} + \sum_{j \in N_G(i), j > i} \vec{m}_{i,j}.$$

of neighbours and m_{ij} is the mask decided between client i and j . Furthermore, it adds r_i , which is a personal mask, for both the paired and the personal mask, secrets based on Shamir's secret sharing method (Shamir1979) are distributed to other clients such that the seed can be reconstructed if sufficient secrets are combined. In case the client drops out, the server can request. By adding the personal mask, the server must choose whether to request the secrets to undo the personal mask, or the secrets to undo the pairwise mask. As the server is not permitted to request both, a malicious server is not able to inspect an individual client's update by requesting the secrets when the client did not actually drop out.

4. When combining all updates on the server, all masks cancel each other out and the global model is reconstructed.

Now, the number of masks and secrets increases rapidly as the number of clients grows, making this only feasible for a limited number of clients, around 10^3 clients.

To extend this further, Secure Aggregation+ (SecAgg+) (Li2021) has been designed, where the concept is the same, but masks and secrets are shared within subgroups of clients, instead of all clients, making the method scale to 10^8 clients. This number of clients is generally sufficient for most federated learning, but in some cases, like experiments with many mobile devices might require a larger number of clients. There exist many other methods further improving secure aggregation like LightSecAgg (So2022) or FastSecAgg (Kadhe2020), where each method improves on specific bottlenecks, such as supporting more clients by limiting computational overhead,

reducing the overhead when a client drops out or supporting models with a large number of parameters.

2.1.1 Using Secure Aggregation in Flower

SecAgg and SecAgg+ are both implemented in Flower through a workflow on the server side and a mod on the client side and can easily be enabled like so on the server side:

```
# Other imports
from flwr.server.workflow import DefaultWorkflow, SecAggPlusWorkflow

# Strategy definition

fit_workflow = SecAggPlusWorkflow(
    num_shares=context.run_config["num-shares"],
    reconstruction_threshold=context.run_config["reconstruction-threshold"],
    max_weight=context.run_config["max-weight"],
)

workflow = DefaultWorkflow(fit_workflow=fit_workflow)

# execute
workflow(grid, context)
```

Figure 1 How to configure Secure Aggregation on the server side in Flower

And like so on the client side:

```
# Other imports
from flwr.client.mod import secaggplus_mod

# Further client_fn definition

app = ClientApp(
    client_fn=client_fn,
    mods=[
        secaggplus_mod,
    ],
)
```

Figure 2 How to configure Secure Aggregation on the client side in Flower

Additional secure aggregation methods can be implemented by users as additional workflow and mods.

2.1.1.1 Performance impact of using Secure Aggregation in Flower

We have created a codebase where Flower is used to train a Resnet18 CNN model (He2016), which is the same model presented during workshop 3 for classifying glaucoma (Wal2025), on the CIFAR10 dataset (Krizhevsky2009). By running the experiment in multiple configurations, such as with and without secure aggregation enabled, we can evaluate the impact of secure aggregation on the performance and runtime of experiments. The results can be found below in Table 1.

Table 1 Impact of Secure Aggregation on model performance and runtime

Configuration	Runtime	Accuracy
Regular training, not federated (1 A100 GPU)	5 min 13 sec	92.77%
Local simulation 5 clients (1 A100 GPU)	15 min 20 sec	88.52%
Local simulation 10 clients (1 A100 GPU)	12 min 06 sec	84.45%
Local simulation 5 clients (1 A100 GPU) + SecAgg+	17 min 9 sec	88.00%
Local simulation 10 clients (1 A100 GPU) + SecAgg+	13 min 50 sec	85.26%
Superlink and 5 Supernodes (clients) on the same node (1 A100 GPU)	49 min 35 sec	88.01%

In the results in Table 1, we can see that SecAgg+ does not impact performance, but does impact runtime by on average 13.05%, as extra communication rounds are required to share the secrets between clients. The experiments with 10 clients have a lower runtime than the experiments with 5 clients because the dataset is divided over 10 clients instead of 5, as all partitions are processed concurrently, the experiment with 10 clients finishes faster. The runtime increases significantly when changing from local simulation to communicating between a Superlink and Supernodes. This has been explored before in Deliverable 2.11 (Wal2025) and the increase observed in these results is similar, and thus further configurations with remote Supernodes in different countries has not been explored, as these can be extrapolated based on earlier work.

2.1.2 Trusted Execution Environments

Trusted execution environments (TEEs) are a secure enclave in the main processor, with increased protections for executed code and loaded data to prevent tampering with code or data and protect data in RAM or transfer from being accessible to other processes. Both major CPU providers, Intel and AMD, provide their own implementations of TEE's, Intel SGX and AMD SEV. When it comes to GPUs, only NVIDIA GPUs starting from the Hopper generation can provide a TEE.

TEEs can provide an alternative to techniques such as SecAgg, as traffic between clients and server is encrypted with TLS and if the code running on the server is confirmed to be safe, processing of the client updates is guaranteed to be safe.

However, the downside of TEEs is that they require specific system settings and generally admin rights are required to utilize the secure enclaves, making them unpractical on shared or managed systems. This requires local experts setting up every client, with inconsistent setups across the federation, whereas SecAgg provides a safe way to aggregate client updates without requiring system specific setups. In addition, to

make the process as a whole secure, TEE's should be set up on all clients as well, which can be challenging given different hardware specifications, or impossible if some clients have hardware that does not support TEEs, which is the case for some GPU's on some clients within the Hereditary project, as described in Deliverable 2.14 (Podareanu2024).

An advantage of TEEs is that client updates are not hidden to the server, which enables more advanced aggregation methods such as FedAdam (Reddi2020) or FedProx (Li2020).

2.2 Differential Privacy

While Secure Aggregation protects individual model updates from being inspected by the central server, a privacy risk remains in the final, aggregated model itself. An adversary could potentially perform inference attacks on the global model over successive training rounds to learn sensitive information about a client's specific data. For instance, they might determine whether an individual with a rare medical condition was part of the training set.

Differential Privacy (DP) addresses this risk by providing a mathematical guarantee of privacy. The core principle of DP is to ensure that the outcome of an analysis is not significantly altered by the presence or absence of any single individual's data in the dataset. In the context of federated learning, it means that the global model would look nearly identical whether your specific data was used for training.

In general, this guarantee could be achieved by adding calibrated statistical noise to the client model updates before they are aggregated. The process then involves two steps on the client side:

- **Gradient Clipping:** The influence of each client's update is limited by clipping its norm to a predefined threshold. This prevents any single data point from having an outsized impact on the model update.
- **Noise Injection:** After clipping, random noise (usually from a Gaussian or Laplace distribution) is added to the model update.

The noise injection gives us a guarantee that is formalized through the concept of privacy budget, denoted by epsilon (ϵ). An algorithm M is considered ϵ -differentially private if, for any two datasets D and D' differing by one person's data, the probability of producing any specific output x is nearly the same. Formally, this is expressed as:

$$\mathbb{P}[M(D) = x] \leq e^{\epsilon} \cdot \mathbb{P}[M(D') = x]$$

A small ϵ (e.g., close to 1) means e^{ϵ} is also small, forcing the output probabilities to be nearly identical whether a specific person's data was used. This provides strong privacy, as an observer of the output cannot confidently infer an individual's participation. In contrast, a large ϵ allows for a much larger divergence in probabilities, implying a weak privacy guarantee where an individual's influence is more visible.

The seminal work, (Abadi2016) introduced the Differentially Private Stochastic Gradient Descent (DP-SGD) algorithm. Their approach operationalized the concepts of clipping and noise injection we saw earlier. For each training step, the algorithm first computes the gradient for every *individual* example in a mini-batch. Then, it clips the L2 norm of each of these individual gradients before averaging them. Only after this averaging step is calibrated (Gaussian) noise added to the result. A key innovation in their work was the Moments Accountant, a more sophisticated method for tracking the cumulative privacy cost over thousands of training iterations. When we train a neural network we can make use of DP-SGD, which adds this noise to the weights of the model.

Algorithm 3 Differentially Private Stochastic Gradient Descent (DP-SGD) Step

Require: Model, batch B , lr η , clipping norm C , noise multiplier σ

```

1: Let  $all\_clipped\_gradients \leftarrow []$ 
2: for all sample  $x_i$  in batch  $B$  do
3:    $g_i \leftarrow \text{ComputePerSampleGradient}(\text{model}, x_i)$ 
4:    $clipped\_g_i \leftarrow \text{ClipGradientNorm}(g_i, C)$ 
5:   Add  $clipped\_g_i$  to  $all\_clipped\_gradients$ 
6: end for
7:  $aggregated\_gradient \leftarrow \text{Average}(all\_clipped\_gradients)$ 
8:  $noise \leftarrow \text{GaussianNoise}(\text{mean} = 0, \text{std\_dev} = C \cdot \sigma)$ 
9:  $noisy\_gradient \leftarrow aggregated\_gradient + noise$ 
10: Update model parameters using  $\eta$  and  $noisy\_gradient$ 
  
```

Figure 3 DP-SGD algorithm (Abadi2016)

This algorithm has two guarantees:

- Any arbitrary computation performed on a private output remains private.
- The privacy budget is cumulative. In DP-SGD, each training step consumes a small portion of the budget, and a Privacy Accountant is used to track the total ϵ spent over the entire training process.

A slightly relaxed definition called (ϵ, δ) -Differential Privacy is often used to make this practical for training. This definition introduces a small parameter, delta (δ), which represents a tiny probability (e.g., 10^{-5}) that the strict privacy guarantee might fail. A randomized algorithm M is (ϵ, δ) -DP if for any neighbouring datasets D , D' and any measurable set S of outputs:

$$\mathbb{P}[M(D) = x] \leq e^\epsilon \cdot \mathbb{P}[M(D') = x] + \delta$$

In an iterative process like DP-SGD, each training step consumes a small amount of the privacy budget. The final ϵ is the cumulative privacy cost over the entire training run (all batches and all epochs). Every interaction with the data increases the total privacy loss.

An algorithm being ϵ -differentially private is equivalent to stating that this random variable is bounded:

$$\|L^{(o)}\|_\infty \leq \epsilon$$

A naive privacy accountancy approach of just adding up the privacy cost, or ϵ , from every single step would result in an overestimated and uselessly large final privacy loss.

We can use a Moments Accountant to treat the privacy loss as a random variable with its own probability distribution. Instead of tracking a single worst-case value at each step, the accountant tracks the entire distribution by computing its Moment-Generating Function (MGF).

In this manner, the accountant builds a more exact picture of the cumulative privacy loss distribution over the entire training process, including the privacy-amplifying effects of random data subsampling. The mathematical justification for the privacy accountant is its ability to leverage the full distributional information of the privacy loss random variable. By composing these distributions exactly using moment-generating functions, it provides a far more accurate and tighter final (ϵ, δ) -DP guarantee compared to older composition theorems, making it feasible to train complex models with meaningful privacy. The reader is encouraged to read (Ponomareva2023) for a full explanation on privacy accounting and the MGF.

A recent example of applying differential privacy in practice for large-scale models comes from Google DeepMind research. In (VaultGemma2025) the authors introduce a 1 billion parameter open source¹ language model fully trained with differential privacy. Nevertheless, the VaultGemma authors note that a utility gap did persists between privately and non-privately trained models. Training a model with a DP process inherently creates a trade-off between privacy and utility. DP models always exhibit measurable drops in accuracy compared to their non-private counterparts. We provide a short demonstration of this in section 2.1.3.

2.2.1 Differential Privacy in Flower

Differential Privacy is implemented in Flower through a workflow on the server side and a mod on the client side and can easily be enabled like so on the server side:

```
from flwr.server.strategy import DifferentialPrivacyClientSideFixedClipping, FedAvg

# Create the strategy
strategy = FedAvg(...)

# Wrap the strategy with the DifferentialPrivacyClientSideFixedClipping wrapper
dp_strategy = DifferentialPrivacyClientSideFixedClipping(
    strategy,
    cfg.noise_multiplier,
    cfg.clipping_norm,
    cfg.num_sampled_clients,
)
```

Figure 4 How to configure Differential Privacy on the server side in Flower

And like so on the client side:

¹ <https://huggingface.co/google/vaultgemma-1b>

```
from flwr.client import ClientApp
from flwr.client.mod import fixedclipping_mod

# Add fixedclipping_mod to the client-side mods
app = ClientApp(
    client_fn=client_fn,
    mods=[
        fixedclipping_mod,
    ],
)
```

Figure 5 How to configure Differential Privacy on the client side in Flower

2.2.2 Performance Impact of using Differential Privacy

There exists a privacy-utility trade-off; stronger privacy guarantees require adding more noise, which typically degrades model performance. The goal is not to eliminate this trade-off, but to manage it effectively by tuning the relevant hyperparameters. On the client-side operations, the adding of noise and clipping operations should introduce a minor computational overhead in each round. However, this is generally negligible, especially if the local model training is a relatively large CNN or transformer model.

We have tested the same Resnet18 model trained on the CIFAR10 dataset with DP as to investigate the impact on runtime and accuracy in a similar fashion to how Secure Aggregation was evaluated, of which the results can be found below in Table 2.

Table 2 Impact of Differential Privacy on model performance and runtime

Configuration	Runtime	Accuracy
Regular training, not federated	5 min 13 sec	92.77%
Local simulation 5 clients (1 A100 GPU)	15 min 20 sec	88.52%
Local simulation 10 clients (1 A100 GPU)	12 min 6 sec	84.45%
Local simulation 5 clients (1 A100 GPU) + DP	15 min 29 sec	88.15%
Local simulation 10 clients (1 A100 GPU) + DP	12 min 3 sec	85.11%

Results shown in table 2 are promising, although it must be noted that the parameters used in differential privacy, the noise multiplier and the clipping norm must be configured and allow a trade-off between performance and privacy. When more strict values are set, training can either take longer or not reach similar performance to an experiment without differential privacy at all. The configuration values differ based on the used models and datasets. Larger models incur larger L2 norms, especially early in training, and non i.i.d. datasets also cause larger L2 norms (Banse2024), as clients will diverge further from the global model. When datasets are distributed, an appropriate mock dataset that mimics the true distribution needs to be available to appropriately configure the differential privacy parameters. Finally, we observe that DP does not add any runtime as adding noise to the model parameters and clipping the model parameters are lightweight operations.

When working with healthcare data, it is common to encounter sequential data. Recently, language models (Transformer models) have gained popularity and proven to be effective at many tasks in text, genomics and image modelling. A common method to improve a language model's performance is by finetuning it on relevant data, which is a sensitive subject in the case of healthcare data, as it is undesirable if the model were to reproduce patient data. As it is not unlikely that transformers might be used within the project, we provide a tangible illustration of the privacy-utility trade-off, we can construct an experiment comparing two models: one trained with standard methodologies and another trained using DP-SGD. We use Shakespeare's works available in the public domain. This experiment is designed to test a model's tendency to memorize specific, unique sequences from its training data. We train a small Llama model from scratch as an example. We inject a few unique sentences within this text which we designate as "canaries." These canaries are used to create prompts for what are as extraction queries; prompts designed to reveal if a model has memorized training data verbatim. For instance, the classically trained model should be able to memorize "Macbeth" which is mentioned a few times across all the works of Shakespeare, while the differentially private model would have a much harder time recalling specific names or places.

Expected Outcomes:

- **Non-DP Model (Memorization):** A standard language model trained on a limited dataset is prone to overfitting. When presented with the first part of a canary sentence, this model is expected to exhibit significant memorization. We expect that it will likely complete the prompt with a verbatim reproduction of the text.
- **DP Model (Generalization):** The core mechanisms of DP limit the influence of any single data point. This discourages the model from learning idiosyncratic details. Consequently, when given the same canary prompt, the DP-trained model will likely fail to complete the sequence verbatim.

We train the transformer decoder and visualize the training curve in the figure below. The privacy-utility trade-off, as visualized in the training curves, is starkly present.

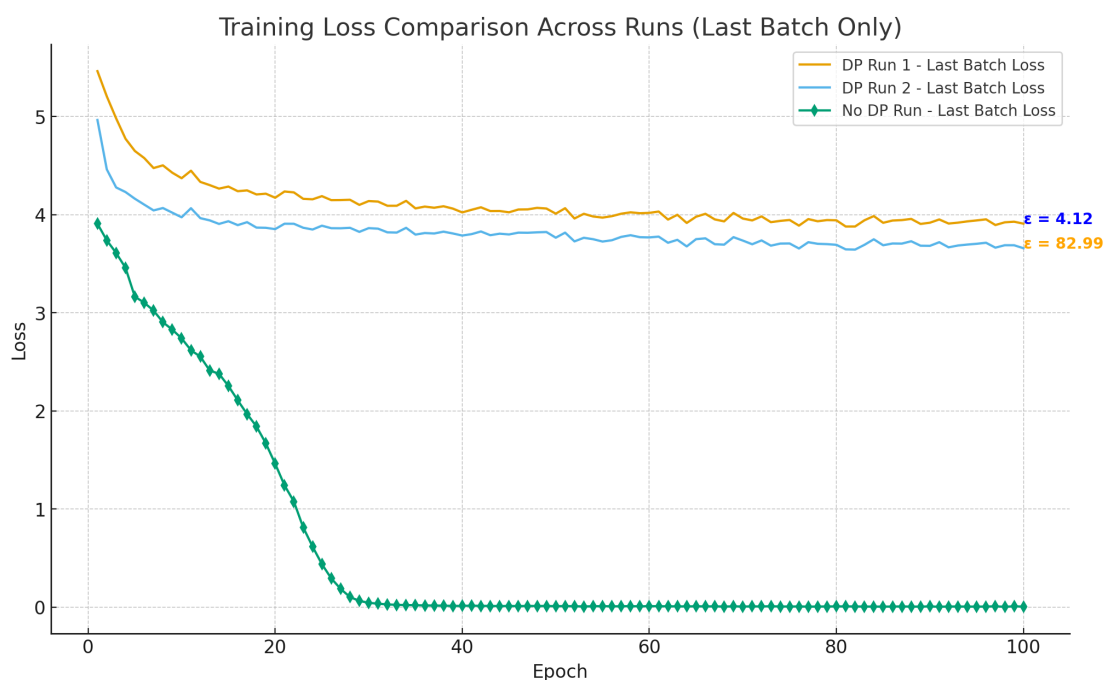


Figure 6 Graph of training loss showing the performance impact of Differential Privacy

In our toy example, we see that the huge quantitative disparity is directly linked to the qualitative findings from the extraction queries; the high terminal loss of the DP-trained models is the statistical representation of their inability to memorize and reproduce specific "canary" training instances. The DP trained model could not reproduce any well-formed sentence. The noise and clipping mechanisms of DP-SGD act as a strong form of regularization, effectively preventing the model from overfitting to unique data points. However, this comes at a substantial cost to model utility. We were not able to generate examples of Shakespearean prose with the differentially private model, while the classically trained model was easily able to overfit on the data. We hypothesize that a large training dataset is required to obtain competitive results, since with DP the individual effect of each datapoint is "blurred".

As the experiment illustrates, finding a set of hyperparameters that yields a model with both a meaningful privacy guarantee (a low ϵ) and acceptable task performance is a non-trivial task.

It should be noted though that in both experiments presented above, a full model was trained on the data, which encompasses a large number of parameters that can be used to encode the data. By utilizing either a pre-trained model where only the last layer(s) are trained or by applying LoRA (Hu2022) to limited layers, the number of parameters that are being optimised can be reduced significantly. By leveraging the general knowledge of an existing model and optimising it for a specific task through limited parameters reduces the need for rigorous noise addition and clipping, potentially increasing the ease of use of differential privacy.

3 Communication within Flower

3.1 Flower messaging API updates

Hereditary Deliverable 2.11 (Wal2025) reported on the then newly introduced Flower Messaging API, which added an additional way to communicate between the server and clients. Initially this could only be done through strategies that follow a specific flow of function calls, whereas the Messaging API enabled the server to perform arbitrarily defined function calls to the clients. In September 2025, Flower version 1.21 was released, in which the strategies are now changed such that Flower utilizes the Messaging API under the hood. This requires a migration for codebases that have been developed using previous versions of Flower, and in Workshop 5 we have covered what is needed for these migrations, for which the Flower team has also released a migration guide².

3.2 gRPC in Flower

The communication within the Flower library is based on gRPC, which is a cross-platform high-performance remote procedure call (RPC) framework. The cross-platform support of gRPC is the backbone that enables Flower to be a multi-platform solution, which for example also supports Android clients. This multi-platform support can also enable large scale federated learning experiments with, for example, data from wearables like smartwatches. All traffic between the server and the clients can be encrypted using TLS, ensuring security in the transmission phase. Ensuring that the TLS communication is safe and cannot be intercepted depends on multiple conditions being met, such as:

- Private and public keys have been exchanged in a safe manner and are stored in a safe manner on all clients. To ensure safety, it is good practice to regularly rotate the certificates, for example, every 90 days.
- Cipher suites are kept up to date to ensure there are no vulnerabilities when encrypting/decrypting the communication. At the time of writing, TLS 1.3 is recommended.
- Firewalls are used to only allow connections from IP ranges that belong to participants in the experiment, to reduce the odds of an outsider abusing an unknown vulnerability
- Errors are handled in a safe manner without revealing any secrets

Operationally gRPC also requires open ports to be available at the side that is receiving communication, in case of federated learning the central server. Flower does not require clients to have open ports, as clients regularly ping the central server to check whether there is any experiment to be ran. The server is required to have 3 open ports for Flower to function.

² <https://flower.ai/docs/framework/how-to-upgrade-to-message-api.html>

4 Conclusion

In this deliverable, multiple techniques have been discussed that apply to different elements of communication within federated learning, each providing additional protection in their domain. Assuming an honest-but-curious server, which means that it follows the protocol, but might attempt to obtain information from anything it receives, Secure Aggregation can be used to protect client updates from being visible to the server, Differential Privacy can be used to reduce training data leakage and by using TLS to encrypt data between the clients and servers, the data is protected from eavesdroppers. Active poisoning, where clients actively try to sabotage the global model have not been discussed, as it is out of scope for communication methods, but this is future work.

References

The bibliographic entries are arranged in lexicographical order based on the key, following the APA style. This enables us to place the entries and citations in the table and text in any sequence, allowing for later sorting while ensuring consistency.

Key	Reference
Beutel2020	Beutel, D., Topal, T., Mathur, A., Qiu, X., Parcollet, T., & Zhao, Y. (2020). Flower: A friendly federated learning research framework. <i>Proceedings of the 2020 IEEE International Conference on Big Data (Big Data)</i> , 1001–1010. https://doi.org/10.1109/BigData50022.2020.9377766
FlowerLabs,n.d.	Flower Labs. (n.d.). <i>Flower: A Friendly Federated AI Framework</i> . https://flower.ai
Bell2020	Bell, J. H., Bonawitz, K. A., Gascón, A., Lepoint, T., & Raykova, M. (2020, October). Secure single-server aggregation with (poly) logarithmic overhead. In <i>Proceedings of the 2020 ACM SIGSAC conference on computer and communications security</i> (pp. 1253-1269). https://doi.org/10.1145/3372297.3417885
Shamir1979	Shamir, A. (1979). How to share a secret. <i>Communications of the ACM</i> , 22(11), 612-613. https://doi.org/10.1145/359168.359176
Li2021	Li, K. H., de Gusmão, P. P. B., Beutel, D. J., & Lane, N. D. (2021, December). Secure aggregation for federated learning in flower. In <i>Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning</i> (pp. 8-14). https://doi.org/10.1145/3488659.349377
Wal2025	Van der Wal, D., Podareanu, D., Rodríguez, J. M., Silvello, G., & Romanovych, A. (2025). Deliverable 2.11: Federated infrastructure design. Zenodo. https://doi.org/10.5281/zenodo.15847728
So2022	So, J., He, C., Yang, C. S., Li, S., Yu, Q., E Ali, R., ... & Avestimehr, S. (2022). Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning. <i>Proceedings of Machine Learning and Systems</i> , 4, 694-720. https://doi.org/10.48550/arXiv.2109.14236
Li2020	Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2020). Federated optimization in heterogeneous networks. <i>Proceedings of Machine learning and systems</i> , 2, 429-450. https://doi.org/10.48550/arXiv.1812.06127
Reddi2020	Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., ... & McMahan, H. B. (2020). Adaptive federated optimization. <i>ICLR 2021</i> https://doi.org/10.48550/arXiv.2003.00295
Abadi2016	Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L. (2016). Deep learning with differential privacy. In <i>Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security</i> (pp. 308-318). https://doi.org/10.1145/2976749.2978318
VaultGemma	Google et al (2025)., VaultGemma: A Differentially Private Gemma Model, https://services.google.com/fh/files/blogs/vaultgemma_tech_report.pdf

Key	Reference
Podareanu2024	Podareanu, D., Dell'Aglia, D., Romanovych, A., & Silvello, G. (2024). Deliverable 2.14: Computing infrastructures. Zenodo. https://doi.org/10.5281/zenodo.14034331
Kadhe2020	Kadhe, S., Rajaraman, N., Koyluoglu, O. O., & Ramchandran, K. (2020). Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. arXiv preprint arXiv:2009.11248. https://doi.org/10.48550/arXiv.2009.11248
Krizhevsky2009	Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.
He2016	He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In <i>Proceedings of the IEEE conference on computer vision and pattern recognition</i> (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90
Ponomareva2023	Ponomareva, N., Hazimeh, H., Kurakin, A., Xu, Z., Denison, C.E., McMahan, H.B., Vassilvitskii, S., Chien, S., & Thakurta, A. (2023). How to DP-fy ML: A Practical Guide to Machine Learning with Differential Privacy. <i>ArXiv, abs/2303.00654</i> .
Banse2024	Banse, A., & Kreischer, J. (2024). Federated learning with differential privacy. https://doi.org/10.48550/arXiv.2402.02230
Hu2022	Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). Lora: Low-rank adaptation of large language models. <i>ICLR</i> , 1(2), 3.

Annexes

Number	Name
1	Workshop 5: Agenda, attendance and detailed explanation of presented work