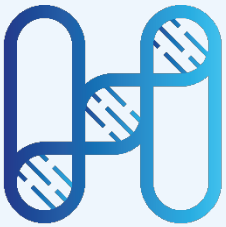

HEREDITARY

HetERogeneous sEmantic Data integratIon for the guT-bRain interplaY

Deliverable 2.14

Computing infrastructures

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

One of the main goals of the HEREDITARY project is establishing secure and scalable computing infrastructures in local medical centres to support federated learning. These infrastructures enable data processing and collaboration without centralising sensitive data. These infrastructures include managed services, computational resources, and storage facilities, ensuring medical centres can actively contribute to the federated learning process while maintaining data privacy.

This deliverable's milestone is setting up the necessary computing and storage infrastructures in medical centres, specifically at UCD, RUMC, UNITO, and UNIPD, and testing the communication protocol that connects these centres to the federated learning infrastructure. This deliverable will show that each medical centre is advancing toward meeting these infrastructure goals and that initial communication protocols have been tested and verified.

Task 2.6 aims to implement the federated learning infrastructure. However, in the context of Deliverable D2.14, the critical objective is to establish computing infrastructures in UCD, RUMC, UNITO, and UNIPD, ensuring that they have the necessary facilities to host federated learning components, process their data locally, and communicate securely with other partners.

We are reporting on the progress in setting up computing infrastructures at UCD, RUMC, UNITO, and UNIPD, including:

- UCD, RUMC, and UNIPD have the required infrastructure in place.
- UNITO is in the process of acquiring the necessary infrastructure.
- Initial communication protocols have been tested between UCD, HES-SO, RUMC, SURF, and UNIPD. Once their infrastructure is ready, further expansion is planned for full collaboration with UNITO.

Additionally, the HES-SO workshop conducted earlier this year successfully demonstrated communication between these medical centres, providing evidence of networking and data-sharing capabilities in a federated learning environment.

DOCUMENT INFORMATION

Deliverable ID	D2.14
Deliverable Title	Computing infrastructures
Work Package	WP2
Lead Partner	SURF
Due date	30.09.2024
Date of submission	27.09.2024
Type of deliverable	OTHER
Dissemination level	PU

AUTHORS

Name	Organisation
Damian Podareanu (Author)	SURF
Daniele Dell'Aglio (Reviewer)	AAU
Anna Romanovych (Contributor)	UNIPD
Gianmaria Silvello (Contributor)	UNIPD

REVISION HISTORY

Version	Date	Author	Document history/approvals
1.0	2/09/2024	Damian Podareanu	Initial deliverable version.
1.1	10/09/2024	Damian Podareanu	Updated references, corrected deliverables and task titles, added additional information about the medical center implementation.
1.1	19/09/2024	Daniele Dell'Aglio	Provided revision of the version and left feedback on the deliverable.
1.2	23/09/2024	Damian Podareanu	Implemented the feedback from AAU and UNITO. Clarified sections 2,3,4. Added missing text and made grammar corrections.
1.3	23/09/2024	Gianmaria Silvello	Overall revision and minor fixes.
1.4	24/09/2024	Anna Romanovych	Corrections of the format and annexes.
1.4	25/09/2024	Daniele Dell'Aglio	Provided feedbacks on the revised deliverables.
1.5	27/09/2024	Damian Podareanu	Added RUMC hardware description, minor formatting and referencing corrections.

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

LIST OF ABBREVIATIONS

Abbreviation	Expanded form
AAU	Aalborg University
AI	Artificial Intelligence
API	Application Programming Interface
BMC	Baseboard Management Controller
Ceph	A distributed storage system
CPU	Central Processing Unit
CIFAR	Canadian Institute for Advanced Research (dataset)
DP	Differential Privacy
ECC	Error-Correcting Code
EU	European Union
EUDAT	European Data Infrastructure
FeTS	Federated Tumor Segmentation
FL	Federated Learning
Flower	A Federated Learning Framework
GDPR	General Data Protection Regulation
GiB	Gibibyte
GPU	Graphics Processing Unit
GPFS	General Parallel File System
HBM2	High Bandwidth Memory 2
HE	Homomorphic Encryption
HES-SO	University of Applied Sciences and Arts Western Switzerland
HIPAA	Health Insurance Portability and Accountability Act
HPC	High-Performance Computing
HPC4AI	High-Performance Computing for Artificial Intelligence
iRODS	Integrated Rule-Oriented Data System
ML	Machine Learning
MLflow	Machine Learning Workflow (Open-source platform for ML lifecycle)
MNIST	Modified National Institute of Standards and Technology (dataset)
NVMe	Non-Volatile Memory Express
PySyft	A Python Library for Secure and Private Deep Learning
RAM	Random Access Memory
RUMC	Radboud University Medical Center
SMPC	Secure Multiparty Computation

Abbreviation	Expanded form
SSD	Solid-State Drive
SURF	Dutch National Supercomputing Center
SFP+	Enhanced Small Form-factor Pluggable
TiB	Tebibyte
UNIPD	University of Padua (Università degli Studi di Padova)
UNITO	University of Turin (Università degli Studi di Torino)
UCD	University of Colorado Denver
US	United States
vCPU	Virtual Central Processing Unit
W&B	Weights & Biases (Machine Learning Experiment Tracking)
WSL2	Windows Subsystem for Linux 2

1 Introduction

One of the main goals of the HEREDITARY project is building a cutting-edge federated learning and analytics infrastructure that leverages secure (super)computing environments to enable distributed machine learning across multiple clients, ensuring data privacy and model security. This infrastructure facilitates collaborative learning while preventing data centralisation, which is particularly important when working with sensitive medical data.

This report describes the current effort in setting up the federated learning infrastructure defined in Task 2.6.

1.1 Overview of the Federated Learning infrastructure

This infrastructure supports data-centric and model-centric federated learning approaches (Jakubik2024), accommodating various stakeholders with differing computational resources and privacy requirements. Clients can process data locally while utilising the computational power of secure supercomputing environments to perform large-scale machine learning on privatised and transferable data. The federated infrastructure employs advanced privacy-preserving techniques, such as differential privacy and homomorphic encryption, ensuring that data confidentiality is maintained throughout the learning process.

The infrastructure also provides a protected communication channel between clients and the central server, enabling secure model updates and ensuring that even clients with fewer resources can participate fully in the training process.

1.2 Objectives of Task 2.6

The primary objective of Task 2.6 is to implement a federated learning infrastructure that is efficient, scalable, and capable of maintaining the privacy and security of data and models across a wide range of clients. To achieve this, several critical infrastructure components have been designed and tested, including:

- Data management to ensure secure data preprocessing, storage, and retrieval while adhering to privacy standards.
- Model management to facilitate efficient, reproducible training, validation, and deployment of machine learning models using frameworks like TensorFlow and PyTorch.
- Communication to establish secure communication between clients and the central server, using technologies such as gRPC to protect data and model transmissions (Beutel2020).
- Resource management to optimises resource allocation, load balancing, and fault tolerance, ensuring that clients with varying computational resources can contribute to the training process (Geyer2017).

The federated learning infrastructure is designed to scale across diverse computing environments, integrating on-premises infrastructure at medical centres with centralised computational resources provided by secure supercomputing platforms like SURF. Privacy-enhancing mechanisms such as differential privacy, Secure MultiParty

Computation (SMPC) (Miller2020), and homomorphic encryption are key to maintaining confidentiality throughout training.

1.3 Verification of milestone M4 - local computing infrastructures

The M4 milestone coincides with deliverable 2.14 from Task 2.6, which details the successful establishment of local infrastructures and testing communication protocols between the UNITO, UCD, RUMC, UNIPD, and SURF centres involved in federated learning. As of month 9, the HEREDITARY project has significantly progressed in developing the federated learning infrastructure. Key achievements include the design of core components, initial testing in sandbox environments, and the development of guidelines for future implementation phases. You can also find details of existing hardware in Annexes 1, 2, 3 and 4.

Certain aspects of the infrastructure still need to be put under active development. Efforts have been made to establish sandbox environments for early testing and to provide guidelines for setting up local infrastructures at medical centres. Communication protocols between medical partners are also being tested to ensure smooth collaboration and data exchange within the federated learning system.

The formal verification of Milestone M4 involves two key objectives:

- Setup of computing infrastructures. This includes the completion of computing and storage setups at the medical centres. These infrastructures are required to support federated learning by enabling local data processing, secure model training, and privacy-preserving computations. Centres like UCD, RUMC, and UNIPD have completed their setups, and UNITO is finalising them. These centres have high-performance servers equipped with GPUs to handle the heavy computational loads required for medical data processing.
- Testing of communication protocols. This ensures seamless data exchange between medical centres and the central federated learning server. This testing was conducted through a workshop, where UCD, AAU, RUMC, HES-SO, SURF, and UNIPD participated in a successful simulation of networking and data-sharing processes. The protocol test suggests the infrastructure facilitates secure, real-time communication during federated learning tasks.

The progress made so far in setting up the infrastructure, as outlined in Deliverable 2.14, verifies Milestone M4. The upcoming focus will be on completing the infrastructure setup at all medical centres, further refining the communication and resource management systems, and continuing to test and integrate advanced privacy mechanisms to ensure compliance with privacy regulations like the GDPR.

Annex 5 presents the codebase to run federated learning methods within the HEREDITARY infrastructure and the code adopted to run the communication tests within the consortium.

This deliverable outlines the progress made in setting up the computing infrastructures at medical centres (UCD, RUMC, UNIPD, and ongoing work at UNITO) and testing the communication protocols that enable secure collaboration between these centres. The infrastructure setup is crucial for allowing the federated learning processes required by

the HEREDITARY project, and future work will focus on expanding the infrastructure's reach and improving its efficiency and security.

2 Infrastructure design document

This section presents the design and implementation of the federated learning infrastructure for the HEREDITARY project. The infrastructure has been designed to balance flexibility, scalability, and security while maintaining compliance with data privacy regulations. The document outlines the core elements of the system architecture, emphasising the integration of key technologies for managing data, models, communication, and resources.

This section provides an overview of the architectural components and covers the current implementation status across participating medical centres, highlighting progress made and ongoing efforts. Finally, it discusses the rationale behind the selected frameworks and technologies, ensuring the infrastructure meets the complex needs of federated learning while maintaining robust security and privacy.

2.1 Architecture overview

The architecture of the HEREDITARY federated learning infrastructure is designed to be modular, flexible, and secure. It consists of several core components, each responsible for different aspects of the federated learning process:

- The data management infrastructure handles data preprocessing, storage, and retrieval, ensuring data privacy and security through technologies like iRODS (iRODS Consortium2021), dCache (Fuhrmann2006), and b2share (EUDAT2021), as well as the built-in capabilities of federated learning frameworks. Adopting a data mesh approach (Fowler2021), the infrastructure decentralises data management by embedding computational policies directly within the federated system (McMahan2017, Ryffel2018), making data management a first-class concern. This enables each data domain to manage its data, ensuring scalability and governance.
- The model management infrastructure supports efficient and reproducible training, validation, and deployment of models using machine learning frameworks like TensorFlow and PyTorch through (self-)hosted tools like MLflow.
- The communication infrastructure ensures secure and efficient communication between clients and the central server, utilising technologies such as gRPC.
- The resource management infrastructure manages load balancing, resource allocation, and fault tolerance, integrating tools like Flower.io and PySyft (Ziller2018).

2.2 Technical implementation progress and choices

This section outlines the progress in implementing the federated learning infrastructure at various medical centres, highlighting the setup status and the technical choices that guide the infrastructure's development. The decisions surrounding framework selection,

server hosting, and communication protocols are discussed, emphasising the project's focus on privacy, scalability, and flexibility. Additionally, this section covers ongoing infrastructure efforts, such as communication tests and integration of advanced privacy-preserving technologies.

2.2.1 Medical centre infrastructure setup progress

This section briefly overviews month 9's progress in implementing local infrastructures.

2.2.1.1 UCD: completed setup

UCD completed the setup of the necessary computing and storage infrastructure, allowing it to contribute fully to the federated learning infrastructure.

The infrastructure includes high-performance computing (HPC) capabilities for local data processing and secure communication channels connecting to the central server.

2.2.1.2 UNIPD: completed setup

UNIPD also completed its infrastructure setup with computational and storage facilities like UCD.

UNIPD participated in communication tests during the HES-SO workshop, confirming that its data can be securely integrated into the federated learning infrastructure.

2.2.1.3 RUMC: completed setup

RUMC is in the process of establishing its infrastructure for federated learning. For local experiments, model development, and analytics, RUMC is utilising several computing resources:

- The DIAG group at RUMC is using the SOL cluster for local experiments and model development. The SOL cluster is designed for medical data processing and machine learning tasks and offers a robust computing environment.
- This cluster is dedicated to fMRI-related experiments, providing RUMC with the necessary computational resources for medical imaging tasks.

For federated learning, RUMC will use an Azure node provided by RTC Data Stewardship, the details of which are still being finalised. Additionally, RUMC plans to utilise the Snellius supercomputer for large-scale federated learning experiments.

RUMC will participate in federated learning communication tests once the infrastructure setup is complete.

2.2.1.4 UNITO: in progress

UNITO is in the process of acquiring the necessary computing and storage infrastructure. Progress has been made to ensure that their local data can be processed securely and they can participate fully in federated learning.

Once the infrastructure is in place, UNITO will join the ongoing communication tests with other medical centres.

2.2.2 Framework selection

We have chosen to implement our federated learning infrastructure using the Flower framework due to its ease of implementation, flexibility, and robust support for various machine learning libraries. Flower's framework-agnostic (Beutel2020) nature allows us to easily integrate with existing codebases and tailor the infrastructure to specific needs.

We considered several frameworks based on model support, aggregation methods, privacy functionality, technical complexity, and maintainability. We analysed Nvidia flare, Flower, Pysyft, Substra, Vantage6, Openfl, FedML, and HeteroFL based on these dimensions.

As a fallback, we have integrated PySyft, a framework known for its robust privacy features, including secure multiparty computation and differential privacy. This fallback provides additional privacy assurances for scenarios requiring heightened security measures.

We've extensively searched multiple frameworks to support this choice, resulting in a score based on support for models, aggregation (Bonawitz2017), privacy, technical complexity , and maintainability.

This choice also sets up the communication process in this case, so gRPC (Flower), since we foresee a need for efficiency and scalability, having the option to fallback onto WebSockets (PySyft) if the need arises for advanced privacy-preserving protocols like SMPC and homomorphic encryption.

2.2.3 Server hosting

The central federated learning server is hosted at SURF, taking advantage of this environment's computational resources, security infrastructure, and connectivity. There are several servers hosted in a virtualised environment. There is a Flower server – client, MLflow, and Wights&Biases (W&B) servers.

The medical centres run their versions of these instances as the deployment process and configuration are available.

2.2.4 Communication protocol testing

We conducted a workshop at HES-SO to test the communication protocols, where UCD, HES-SO, SURF, RUMC, and UNIPD participated in networking and data-sharing simulations. This workshop demonstrated that the medical centres could securely exchange model updates and data within the federated learning framework.

Although UNITO was separate from the initial workshop due to ongoing infrastructure acquisition, they will be included in future communication tests once their setup is complete.

Similarly, we will investigate integration with RUMC cloud-based resources, which, although supported by our current infrastructure solution, wasn't tested during the workshop.

3 Adapting the codebase for Federated Learning

In this section, we delve into adapting existing machine learning codebases for federated learning. We compare a vanilla federated learning implementation with a customised solution tailored for specific applications like the FeTS (Federated Tumor Segmentation) project. We used the FETS project to illustrate code changes needed and tradeoffs. This knowledge will be applied to the codebase produced by the Hereditary consortium. Additionally, we provide a step-by-step guide to assist developers in modifying their codebases to support federated learning, ensuring compatibility, efficiency, and privacy compliance.

3.1 Vanilla codebase vs. customised implementation

Adapting a codebase for federated learning can involve using a standard, out-of-the-box implementation or developing a highly customised solution that meets specific project requirements.

3.1.1 Vanilla Federated Learning implementation

The vanilla implementation of federated learning using Flower involves basic server-client communication. Each client processes its data locally and sends updates to the central server.

The medical centres with infrastructure in place (UCD, UNIPD) are currently operating with a simple version of this setup to ensure communication channels are secure and data is processed correctly before advancing to more complex tasks.

This implementation involves setting up a central server that coordinates training across multiple client nodes, each of which processes its data locally. The basic steps are as follows:

- The server orchestrates the training process. It initialises the model, receives model updates from the clients, aggregates these updates, and sends the aggregated model back to the clients. (This is server.py in the GitHub codebase)
- Each client trains the model on its local data and periodically sends updates to the server. The client code is generally straightforward, focusing on loading the data, performing local training, and communicating with the server. (This is client.py in the GitHub codebase)

3.1.2 Customised FeTS implementation

We implemented a solution for the FeTS (Federated Tumor Segmentation) project to validate the infrastructure beyond basic tutorial support. This requires handling specialised medical data. We ensured the infrastructure could support complex model architectures and enhanced privacy features. We also provide these changes for the toy example for the Medical MNIST dataset. This involves customising the data loading and preprocessing steps to ensure compatibility with the federated learning setup. (This is fets_data_provider.py in the GitHub codebase)

The models used in FeTS are tailored for medical image analysis and require adjustments to the training and evaluation processes. (The entire codebase is in the

vit_example; additional examples are provided in the central repository through the *_medical_mnist.py examples.)

3.2 Adapting a codebase for Federated Learning

This section provides a step-by-step methodology for adapting a non-federated learning codebase to meet specific project needs, including data handling, model customisation, and privacy integration.

Step 1: Analyze requirements

- Evaluate the data characteristics and understand the nature of your data, including format, size, and preprocessing needs.
- Asses model complexity and whether your current model architecture suits federated learning or needs adjustments.
- Identify privacy needs, determining the level of confidentiality required based on regulations and data sensitivity.

Step 2: Select an appropriate Federated Learning framework

For framework evaluation and selection, compare Flower, PySyft, and NVIDIA FLARE based on compatibility, scalability, and privacy features. For instance, Flower offers flexibility and ease of use, while PySyft provides advanced privacy mechanisms.

Step 3: Modify Data Handling Procedures

- Local data processing ensures that all data preprocessing and augmentation occur locally for each client to maintain data privacy.
- Adapt data loaders to handle local datasets, considering variations in data formats and structures across clients.
- Implement strategies to address non-independent and identically distributed (non-IID) data scenarios.

Step 4: Adapt the model architecture

- Model refactoring: Modify the model to be compatible with federated learning, which may involve layers, activation functions, or parameter initialisation changes.
    - Batch Normalization layers rely on batch statistics, which can vary significantly across clients due to non-IID (Independent and Identically Distributed) data. This can lead to model divergence. BatchNorm layers can be fixed by replacing them with GroupNorm or LayerNorm layers, which use instance-level statistics instead of batch-level statistics.
    - Clients with limited computational resources may need help with complex models. Reducing the number of layers, neurons, or filters can be beneficial. Use depthwise separable convolutions or reduce the size of fully connected layers.
    - Use activation functions that are efficient and widely supported, like ReLU, Leaky ReLU, or ELU.
    - Convergence must ensure all clients start training using the same initial model parameters. It is advisable to use a common initialisation method and, if necessary, a fixed random seed.

    - Some federated learning frameworks may have specific requirements for model definitions. In Hereditary, we decided that Flower and PySyft would cover all our existing and future models.
    - It's important to ensure your model does not rely on global states that could differ between clients. So, it's advisable to encapsulate all model-related variables within the model class or client instance.
- Model serialisation: Implement serialisation methods for transmitting model parameters between the server and clients. For PyTorch this would be *state_dict()* and *load_state_dict()*, while for TensorFlow they are *model.get_weights()* and *model.set_weights()*
    - We can compress the data through libraries like gzip, zlib, or lz4 for improved serialisation performance.
    - If your model includes custom layers or data types, ensure they are serialisable.
- Resource optimisation: Optimize the model for performance across clients with different computational capabilities. This can be done by applying 16-bit (FP16) or 8-bit (INT8) quantisation, eliminating weights that contribute least to the model's performance, choosing models designed for efficiency like MobileNet, SqueezeNet, or EfficientNet, or by using knowledge distillation.

Step 5: Implement Federated Training logic

Please revisit section 3.1 for a practical implementation of the training logic. The general steps are straightforward:

- Develop a training loop within the federated learning framework, handling local epochs and batching.
- Customise the server's aggregation method (e.g., FedAvg, FedProx) based on the project's convergence and performance requirements.
- Utilise secure and efficient communication protocols like gRPC or WebSockets as your chosen framework supports.

Step 6: Integrate privacy-preserving techniques

This complex step will be addressed in more detail in the following section (section 4). The techniques we have currently in scope for Hereditary are:

- Implementing noise mechanisms to model updates before sending them to the server.
- Defining and monitoring the privacy budget (epsilon, delta) to balance privacy and model utility.
- Dividing model parameters into shares distributed among clients or secure servers.
- Ensuring computations are performed without decrypting the data. This is done by applying encryption that allows computations on ciphertexts, producing an encrypted result that, when decrypted, matches the result of operations performed on the plaintext.

4 Incorporating privacy mechanisms in Federated Learning Infrastructure

Privacy is a fundamental concern in the HEREDITARY project because we deal with sensitive medical data. Federated learning (FL) allows model training across multiple decentralised devices or servers holding local data samples without exchanging them. However, even in FL, unintended information leakage through model updates is risky. To mitigate these risks, we integrate various privacy-enhancing techniques into our federated learning infrastructure (Kairouz2021). These techniques include differential privacy (DP), secure multiparty computation (SMPC), and homomorphic encryption (HE). They help ensure compliance with data protection regulations like GDPR and HIPAA, safeguarding patient confidentiality throughout the model training process.

4.1 Introduction to privacy in Federated Learning

Federated learning enables collaborative model training while keeping data localised, essential for maintaining sensitive medical records' privacy. Despite this advantage, model updates transmitted between clients and servers can inadvertently leak information about the underlying data. To address this concern, we incorporate advanced privacy mechanisms into our FL infrastructure, ensuring that individual data points cannot be reconstructed or inferred from shared information.

4.2 Implementing Differential Privacy using Flower

Differential privacy (DP) provides a mathematical framework to quantify and control the privacy loss incurred when computing sensitive data. By adding controlled noise to the model updates, DP ensures that the inclusion or exclusion of a single data point does not significantly affect the computation's output, thereby protecting individual privacy.

Implementing DP in federated learning involves several critical steps:

1. Understanding the complexity of setting privacy parameters

   Setting appropriate values for the privacy parameters epsilon (ε) and delta (δ) is crucial but challenging. These parameters control the trade-off between privacy and model utility:

   - Epsilon (ε) represents the privacy budget. A smaller ε provides stronger privacy guarantees but may reduce model accuracy.
   - Delta (δ) represents the probability that the privacy guarantee does not hold. It is usually set to be less than the inverse of the dataset size.

   Determining suitable values for ε and δ involves:

   - Data sensitivity assessment involves evaluating the sensitivity of the medical data, considering factors like potential harm from disclosure and data uniqueness.
   - Consulting legal requirements and guidelines may specify acceptable ranges for ε and δ.

- Investigate using methods like the Privacy Loss Distribution (PLD) or Moments Accountant to compute the cumulative privacy loss over multiple training iterations.
- Conducting experiments to observe the impact of different ε and δ values on model performance and privacy, aiming for a balance that meets project requirements.

2. Integrating DP into the training process

   To incorporate differential privacy into our federated learning system using Flower, we modify the client-side code to add noise to the model updates. Flower supports DP through its built-in mechanisms, allowing clients to perturb their updates before sending them to the server. An example implementation can be found in the mock_dp.py file in the central repository.

```
# Example using PyTorch and Opacus library
import torch
from opacus import PrivacyEngine

model = ...  # Your model
optimizer = ...  # Your optimizer
data_loader = ...  # Your data loader

privacy_engine = PrivacyEngine(
    model,
    batch_size=batch_size,
    sample_size=len(data_loader.dataset),
    alphas=[10, 100],
    noise_multiplier=noise_multiplier,
    max_grad_norm=max_grad_norm,
)
privacy_engine.attach(optimizer)
```

3. Configuring the server for DP

   On the server side, we ensure that the aggregation process accommodates the differentially private updates from the clients. The server must coordinate and enforce differential privacy across all clients by:

- Ensuring all clients adhere to the agreed-upon DP parameters.
- Using techniques that prevent the server from inferring individual client updates.

4. Monitoring and adjusting privacy levels

It is essential to track the cumulative privacy loss throughout training and adjust parameters as needed. This involves keeping a record of the privacy budget consumed during training and changing the noise multiplier based on training progress and privacy loss.

4.3 Exploring additional privacy mechanisms

While differential privacy is a key component of our privacy strategy, we are also exploring other techniques to enhance security in specific scenarios.

Secure multiparty computation (SMPC)

Secure multiparty computation allows multiple parties to jointly compute a function over their inputs while keeping those inputs private. Using SMPC, we can calculate shared data without exposing the underlying values. We utilise PySyft to facilitate SMPC in our infrastructure. For example, model parameters can be split into shares and distributed among parties. Operations are then performed on these shares without revealing the underlying data.

```
# PySyft SMPC example
import syft as sy
# Assume alice and bob are virtual workers
alice = sy.VirtualWorker(hook, id="alice")
bob = sy.VirtualWorker(hook, id="bob")
# Secret sharing the data
shared_data = data.share(alice, bob)
# Operations can now be performed on shared_data
```

Homomorphic encryption (HE)

Homomorphic encryption allows computations to be performed directly on encrypted data without decryption. Although we have yet to fully integrate HE, we are investigating its potential use to enhance privacy in our federated learning system. We can ensure that sensitive information remains confidential even during processing by applying encryption schemes like Paillier (Paillier1997) or CKKS (Cheon2017) to model updates.

However, HE introduces challenges such as computational overhead and implementation complexity. We assess the feasibility of integrating HE without significantly impacting training times or disrupting existing machine learning frameworks.

4.4 Practical considerations and challenges

Implementing privacy-preserving techniques in federated learning brings several practical challenges that must be addressed to ensure the success of the HEREDITARY project.

Techniques like DP, SMPC, and HE can introduce computational and communication overheads. For instance, adding noise in DP or performing cryptographic operations in SMPC and HE increases the computational load on clients and servers. This can be particularly challenging for clients with limited hardware capabilities. These issues can be mitigated by utilising efficient algorithms and libraries optimised for performance, taking advantage of them, and distributing the computational load where possible to improve efficiency. In the case of DP, increased computational requirements may limit hardware availability. We are carefully balancing the level of privacy with the computational resources required. As participating medical centres grow, maintaining efficiency and privacy becomes more complex. Scalability challenges include managing increased communication overhead and ensuring consistent performance across diverse hardware environments. We plan to address scalability by implementing hierarchical federated learning and organising clients into clusters or groups to reduce the load on the central server and improve communication efficiency.

Our privacy measures are designed to comply with data protection regulations such as GDPR and HIPAA. Ensuring legal and ethical compliance is essential for the project's success.

To achieve this, we are:

- Conducting legal reviews by consulting with legal teams to validate compliance with relevant regulations.
- By seeking guidance on ethical considerations related to data usage and privacy.

4.5 Federated Analytics with Flower and PySyft

In this section we will briefly describe our approach to achieve federated analytics through Flower and PySyft.

4.5.1 Federated Analytics using Flower

Federated analytics allows for data analysis across distributed datasets without centralising the data. Using Flower, federated analytics can be achieved by leveraging the framework’s flexible aggregation methods.

An example can be found in federated_analytics_client.py.

- Each client loads local data using the FetsDataProvider.
- Clients perform local analysis (in this case, calculating the mean of a feature column) and send aggregated results (e.g., the regional mean) to the server.
- The server collects local analytics results from clients and aggregates them using a weighted average.

Federated analytics does not involve training; it focuses on collecting aggregated data insights without sharing the raw data.

4.5.2 Federated Analytics using PySyft

PySyft enables secure federated analytics through privacy-preserving techniques like SMPC. This allows data to be analysed and decentralised while keeping individual data points secure.

- An example can be found in the federate_analytics_client_pysyft.py.
- The encrypted sums are aggregated to compute a global sum and global mean, ensuring no client's raw data is exposed.
- The final result is decrypted and returned in clear text for the aggregated result.

This implementation ensures secure federated analytics through PySyft's SMPC, maintaining data privacy while enabling data aggregation.

4.6 Future work and enhancements

We plan to continue exploring advanced privacy techniques and rigorously test our implementations to ensure robust security and compliance.

5 Setting up the Federated Learning infrastructure on your local computer

5.1 Introduction

Setting up the federated learning infrastructure on a local machine allows you to experiment with the codebase, run simulations, and contribute to the project's development. To get everyone started, we had some online consultations and conducted an in-person workshop with the consortium members, using both EU and US-based servers to ensure broad accessibility and collaboration.

This section provides a step-by-step guide to setting up the infrastructure locally, including running the code using Docker for easy deployment. The codebase is available on GitHub, and we will reference specific repositories and files that you will need to clone and execute.

5.2 Prerequisites

Before setting up the environment, ensure that your system meets the following prerequisites:

- **Operating System:** Linux, macOS, or Windows with WSL2
- **Python:** Version 3.7 or higher
- **Docker:** Installed and running (Docker Desktop for Windows and macOS)
- **Git:** Installed and configured for cloning repositories

5.3 Cloning the repository

Begin by cloning the Hereditary project repository from GitHub:

```
git clone https://github.com/sara-nl/Hereditary.git
```

This repository contains all the necessary code for the federated learning setup, including server and client implementations and specific datasets like CIFAR and Medical MNIST.

5.4 Running the Federated Learning setup with Docker

We have provided Docker configurations to simplify the setup process. Docker containers ensure that the environment is consistent across different systems, reducing the chances of compatibility issues.

1. Navigate to the root directory of the cloned repository and run

   docker-compose build

   This command creates the necessary Docker images for the server and client components and pulls them in depending on the environment.

2. To start the server, run the following command:

   docker-compose up server

   This launches the federated learning server using Flower, configured to handle multiple client requests.

3. To start one or more clients, use the following command:

   docker-compose up client

   You can run multiple client instances by repeating this command in separate terminal windows or by scaling the Docker service:

   docker-compose upscale client=3

   This command starts three client instances connected to the server for federated learning.

4. Docker Compose displays logs in the terminal, allowing you to monitor the server and client interactions. You can also use Docker's built-in commands to view logs for specific services: `

   docker logs <service_name>

5.5 Running inference

Once the training is complete, you can run inference on the trained models locally:

1. The trained model will be saved on the server container. You can access it using Docker's file system commands or configuring a shared volume during the setup.

2. Use the following command to run the inference script:

   docker-compose run client python inference.py

   This script loads the trained model and run inference on a test dataset.

5.6 Adjusting the codebase

You can directly modify the files in the cloned repository if you need to adjust the codebase to work with different datasets or models. Critical files you might want to edit include:

- **client.py** and **server.py** for basic federated learning setup
- **client_medical_mnist.py** and **server_medical_mnist.py** for specialised datasets
- **flower_tutorial.py** for examples and further customisation

After making changes, rebuild the Docker images and restart the services:

docker-compose build && docker-compose up

5.7 Workshops and consortium collaboration

We have conducted two workshops to help consortium members set up and start working with the federated learning infrastructure.

The first workshop was held in presence at HES-SO on 16 and 17 May 2024 with representatives from HES-SO, UNIPD, UCD, SURF, RUMC, AAU and ONTO. The second workshop was held online on 1 July 2024.

These workshops covered:

- Introduction to Federated Learning: Basic concepts and the rationale behind using frameworks like Flower and PySyft.
- Setting up the environment: This section provides step-by-step guidance on setting up the environment, cloning the repository, and running the code locally.
- Running distributed training: Demonstrations on how to run federated learning across EU and US-based servers, ensuring cross-border collaboration.
- Hands-on sessions: Practical sessions where participants configured their environments, ran the provided code, and contributed to the project.

6 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. *Proceedings of the 2020 IEEE International Conference on Big Data (Big Data)*, 1001–1010. https://doi.org/10.1109/BigData50022.2020.9377766
Bonawitz2017	Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., & Song, S. (2017). Practical secure aggregation for federated learning on user-held data. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (CCS '17), 1175–1191. https://doi.org/10.1145/3133956.3133982
Cheon2017	Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi & T. Peyrin (Eds.), Advances in Cryptology—ASIACRYPT 2017 (pp. 409–437). Springer. https://doi.org/10.1007/978-3-319-70694-8_15
EUDAT2021	EUDAT. (2021). *B2SHARE: EUDAT's secure, reliable and trusted service for storing and sharing research data*. https://b2share.eudat.eu
Fowler2021	Fowler, M. (2021). Data mesh: Principles and logical architecture. *martinfowler.com*. Retrieved from https://martinfowler.com/articles/data-mesh-principles.html
Fuhrmann2006	Fuhrmann, P., & German, V. (2006). *dCache, Storage System for the Future*. European Organization for Nuclear Research (CERN). https://dcache.org
Geyer2017	Geyer, R. C., Klein, T., & Nabi, M. (2017). Differentially private federated learning: A client level perspective. *Proceedings of the 2017 IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT '17)*, 216-227. https://doi.org/10.1109/PASSAT.2017.109
iRODS Consortium2021	iRODS: Integrated Rule-Oriented Data System (Version 4.2.10) [Software]. https://irods.org
Jakubik2024	Jakubik, J., Vössing, M., Kühl, N., Walk, J., & Satzger, G. (2024). Data-centric artificial intelligence. *arXiv preprint* arXiv:2212.11854v4. https://doi.org/10.48550/arXiv.2212.11854
Kairouz2021	Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., & Zhao, S. (2021). Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1), 1–210. https://doi.org/10.1561/2200000083

Key	Reference
McMahan2017	McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (AISTATS '17), 1273-1282. https://arxiv.org/abs/1602.05629
Miller2020	Miller, A., Fritsch, L., Jepsen, T. H., Williams, J. E., & Guo, M. (2020). Secure multi-party computation in federated learning. *Proceedings of the 2020 IEEE International Conference on Big Data (Big Data)*, 3197–3204. https://doi.org/10.1109/BigData50022.2020.9377857
Paillier1997	Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In J. Stern (Ed.), Advances in Cryptology—EUROCRYPT '99 (pp. 223–238). Springer. https://doi.org/10.1007/3-540-48910-X_16
Ryffel2018	Ryffel, T., Trask, A., Dahl, M., Wagner, B., Mancuso, J., Rueckert, D., & Passerat-Palmbach, J. (2018). A generic framework for privacy-preserving deep learning. *arXiv preprint arXiv:1811.04017*. https://arxiv.org/abs/1811.04017
Ziller2018	Ziller, M., Peters, A., & Wagner, M. (2018). Secure aggregation for federated learning using PySyft. *Proceedings of the 2018 International Conference on Privacy, Security, Risk and Trust (PASSAT '18)*, 412-422. https://doi.org/10.1109/PASSAT.2018.153

7 Annexes

Number	Name
1	UNIPD Medical Server (Hardware Overview)
2	UNITO HPC4AI System (Hardware Overview)
3	SURF Snellius Supercomputing System (System Overview)
4	RUMC computational resources (Hardware Overview)
5	Codebase for the execution of federated learning methods