Substra: a framework for privacy-preserving, traceable and collaborative Machine Learning

Mathieu Galtier[1, 2]
mathieu.galtier@owkin.com

and

Camille Marini[1, 2]
camille.marini@owkin.com

[1]Owkin, France
[2]Substra Foundation, France

October 25, 2019

Machine learning is promising, but it often needs to process vast amounts of sensitive data which raises concerns about privacy. In this white-paper, we introduce Substra, a distributed framework for privacy-preserving, traceable and collaborative Machine Learning. Substra gathers data providers and algorithm designers into a network of nodes that can train models on demand but under advanced permission regimes. To guarantee data privacy, Substra implements distributed learning: the data never leave their nodes; only algorithms, predictive models and non-sensitive metadata are exchanged on the network. The computations are orchestrated by a Distributed Ledger Technology which guarantees traceability and authenticity of information without needing to trust a third party. Although originally developed for Healthcare applications, Substra is not data, algorithm or programming language specific. It supports many types of computation plans including parallel computation plan commonly used in Federated Learning. With appropriate guidelines, it can be deployed for numerous Machine Learning use-cases with data or algorithm providers where trust is limited.

Context Substra is an open source framework which can be found on the Substra github (https://github.com/SubstraFoundation). It was originally developped by Owkin (https://www.owkin.com/), which proposes an enterprise version of Substra for Healthcare. It is now hosted by the nonprofit Substra Foundation (https://www.substra.ai/).

1 Introduction

Machine Learning (ML) is a promising field with many applications; organizations of all sizes are practising it, from individual researchers to the largest companies in the world. In doing so, they concentrate an extremely large amount of data. Today, data business is flourishing. However, these practices raise important ethical questions which ultimately could limit the potential social benefits of ML [14, 25]. ML requires large amounts of data to learn from examples efficiently [18]. In ML more data often leads to better predictive performance. Usually, different sources, such as users, patients, measuring devices etc, produce data in a decentralized way. This source distribution makes it difficult to have enough data for training accurate models. Currently, the standard methodology for ML is to gather data in a central database.

However, data is often sensitive. In the case of personal data, which are explicitly related to an individual, privacy is at stake. Personal data are particulary useful and valuable in the modern economy. With personal data it is possible to personalize services, which has brought much added value to certain applications. This can involve significant risks if the data are not used in the interest of the individual. Not only should personal data be secured from potential attackers, but the organisations collecting data should also be transparent and aligned with user expectations. In the European Union, the General Data Protection Regulation (GDPR) [3] has imposed consent and control of citizens over their personal data as a fundamental right. Beyond privacy, data can also be sensitive when it has economic value. Information is often confidential and data owners want to control who accesses it. Examples range from classified information and industrial secrets to strategic data which can give an edge in a competitive market. From the perspective of tooling, privacy-preserving and confidentiality-preserving are very similar and differ mostly in the lack of regulation covering the latter.

Thus, a tradeoff exists between predictive performance improvement versus data privacy and confidentiality. ML always needs more data, but data tend to be increasingly more protected. The centralization paradigm where a single actor gathers all data on its infrastructure is reaching its limit.

A relevant way to solve this tradeoff lies in **distributing computing** and remote execution of predictive tasks. In this approach, the data themselves never leave their nodes. In ML, this includes **Federated Learning**: each dataset is stored on a node in a network, and only the algorithms and predictive models are exchanged between them [22, 12]. This immediately raises the question of the potential information leaks in a trained model. The research on ML security and privacy has seen a significant increase in recent years covering topics from model inversion [16] and membership attacks to model extraction [27]. A residual risk is that data controllers still have to trust a central service that orchestrates federated learning, and distributes models and metadata across the network. Research in Secured Multi Party Computation (SMPC) [17] has proposed several schemes and tools to solve the problem and some have been recently proposed precisely on the ML context of this whitepaper [20]. The results are promising, but a large computing and communication overhead may slow the growth of this field.

Reliability and reproducibility of ML is also a major challenge to wide social and market adoption. It is now clear that ML is relevant in many well defined industrial applications, but it is restricted to standardized tasks and still needs to improve in the face of the inherent variability of certain phenomena. For some sensitive applications, such as Healthcare, one can not tolerate mistakes. In ML, predictive models are trained from a set of examples and, even with the best technology, models will perform poorly on a new example which may be significantly different from the training dataset. Building representative datasets is key to creating robust models. Consequently, it is fundamental to consider the training of predictive models together with sound evaluation. A sound evaluation should always be performed on a representative test dataset of the target population of data. This evaluation should be entirely traceable and reproducible. Furthermore, in sensitive situations, evaluation should be done by independent organisations. Today, there is a lack of collaborative tools which could make the evaluation or certification processes regarding the ML predictions more reliable.

Parallel to the growth of ML and its risk mitigations, the field of **Distributed Ledger Technologies (DLT)** has recently gained momentum with the rapid rise of blockchain technology from early conception to deployment of mature technology; networks that are both broadly used and indestructible. Services built on top of blockchains are said to be trustless: one does not need to trust a third party to guarantee integrity and availability of the service. Today, a large number of users contribute daily to the secure hosting of a distributed and unfalsifiable database, called a ledger, on networks powered by protocols such as Bitcoin [23] or Ethereum [28]. Ledger networks are often operated through smart-contracts which are simply traceable functions on the state of the ledgher. This amounts to creating *trustless* services where one does not rely on a third party to provide a reliable service. In the wake of public blockchains, several private blockchain frameworks have emerged, many of them are hosted by the Hyperledger initiative [5, 11]. The core difference is that private blockchains are deployed within a restricted group of users. This significantly simplifies the underlying consensus mechanism and, in particular, removes the requirement of large computing power associated with the Proof of Work consensus mechanism, as is required by the Bitcoin network.

In this whitepaper, we describe **Substra, a traceable and privacy-preserving framework for collaborative ML** which tackles the challenges of robust ML on sensitive data. It orchestrates the remote execution of ML models over distributed datasets under advanced privacy constraints. Substra relies on a private DLT to implement distributed learning in a trustless way. It connects several users controlling different datasets, to algorithms providers and independent performance evaluators. In this document, we cover the principles, concepts, usage, architecture, ML orchestration features and risk analysis of the proposed technology.

2 Principles

Three core principles drive the development of Substra:

- **Collaboration**. In practice, data is often spread among several partners and the algorithmic expertise can belong to yet another institution. Substra is rooted in the belief that state of the art ML will be built within networks of partners, in particular when the data is sensitive.

- **Privacy**. Data controllers should never expose their data to obtain a service based on ML. Sensitive data should remain private and never be transferred to a third party. Favoring remote execution rather than remote access, Substra is decentralized and makes it impossible for anyone but the owner or authorized algorithms to access the data.

- **Traceability**. Complete traceability of all ML operations is essential not only to guarantee privacy of data, but also to provide an untampered history of the training of any predictive model. This is necessary to support any reliability claim regarding the performance of a model.

3 Concepts

Here, we introduce the main concepts underlying Substra. In fact, Substra is a framework to orchestrate computations in different nodes over several *Assets* under the constraint of explicit *permission regimes.*

3.1 Nodes

Nodes are standalone computing and storage resources running the Substra code. They are organised into a network. It is assumed that independent partner organizations control their respective nodes. They form a private network, where every node is connected to all others.

In Substra, users are authenticated through the node they belong to (see section 5 on architecture). There are only credentials at the institution level: individual users are not personally identified at the network level. Thus, throughout the document, we will refer to users, organisations, institution, partners or, nodes indistinctly.

3.2 Assets

Substra registers, stores and organizes computations on four different kinds of *Assets*: Objectives, Datasets, Algorithms and, Models. These assets can be private or shared depending on their *permission regime.*

- An *Objective* clearly defines the purpose of the computations. It specifies (i) the data format that the *Dataset*, *Algorithm* and *Model* must follow, (ii) the identity of the test data points

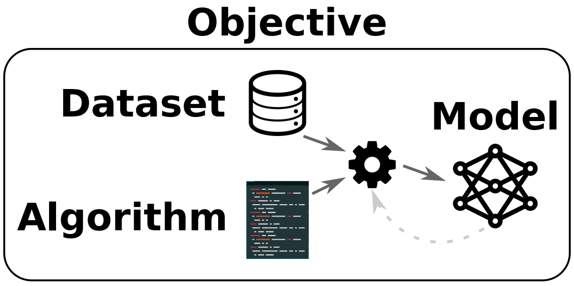


Figure 1: The four types of assets in Substra.

used to compare and evaluate the models and, (iii) the metric calculation script which is used to quantify the accuracy of a model.

- A *Dataset* aggregates numerous data points under a common standard format. It includes a single *Opener* script which imports and opens the file using libraries specific to the data type.
- An *Algorithm* is a script which specifies the method to train a *Model* on a *Dataset*. In particular, it specifies the model type and architecture, the loss function, the optimizer, hyperparameters and, also identifies the parameters that are tuned during training.
- A *Model* is a potentially large file containing the parameters of a trained model. In the case of a neural network, a model would contain the weights of the connections. It is the result of training an *Algorithm* with a given *Dataset*. In Substra a *Model* is defined through a training task which is specified by a *traintuple*. The *Model* can be evaluated via the defintion of a *testtuple*.

Substra is agnostic with respect to the *Assets* nature and format. Substra can be used in any field for any **supervised Machine Learning problem**. Consequently, it is up to the users to design and respect consistent interoperability conventions for their specific application. *Openers* and *Algorithms* have to be manually made compatible for each *Objective*. The format of each *Asset* provided by users must therefore be documented in detail.

3.3 Permission regimes

Each *Asset* in Substra has its own *permission regime*. A *permission regime* specifies which organizations can process or download a given *Asset*, as described below.

There are two types of permissions:

- The permission to **process** an *Asset* provides the ability to utilize it in a training or prediction task. If permission to process an *Asset* is given to a node, then the latter can request the processing of the *Asset*. But the *Asset* never leaves the node of its owner: the processing is done within the owner node. For instance, a *Dataset* can be used to train a model by any organization having the process permission.
- The permission to **download** an *Asset* provides the ability to retrieve and access the *Asset*. If permission to download an *Asset* is given to a node, then the *Asset* will be shared between this node and the owner node. Of course, the point of Substra is that no *Dataset* is ever given a download right (excluding samples of anonymized data points for prototyping).

In Substra, having the download right over an asset implies having the process right.

Beyond the whitelist of organizations having process and/or download rights, an *Asset* can also be made available for processing only for specific purposes. To do so the *Asset* owner must provide a whitelist of *Objectives* for which the *Asset* can be used.

Models that are created by Substra inherit their *permission regime* from the *Assets* that were used during its creation. By default, the process and download whitelists of the new model are the intersection of the whitelists of the *Dataset*, *Algorithm* and initial *Models* used for training. For now, at least one organization must be given the download right so as to be able to store the *Asset*. This choice is made explicitly in the *traintuple* specifying the *Model* creation.

In Substra, the permission regimes are enforced a priori: a *traintuple* can only be created if it respects the *permission regime* of each *Asset* involved. In other words, computations can only be triggered by an organization having at least the process right over all assets referenced in the *traintuple*. In fact, the permissions are implemented in trustless smart-contracts which filter the addition of *traintuples* in the ledger.

3.4 Computations

At its core, Substra is a tool to orchestrate the execution of training tasks. These training tasks turn a triplet of *Dataset*, *Algorithm*, and *Model* into an updated *Model* (see figure 1). The goal is to fit the model to a new set of data in order to increase performance on similar data points. The specification of a training task is entirely contained in a *Traintuple*, which gathers the relevant information about the necessary *Assets* and all technical variables to unequivocally describe a training task.

Traintuples have a counterpart for the test of a *Model* on a separate *Dataset*: *Testtuples*. They correspond to the specification of evaluation tasks of *Models* resulting from *Traintuples*.

One can form a chain of training (and evaluation) tasks, where a model is sequentially updated with various *Datasets* and/or *Algorithms*. We call such chains of tasks *Compute Plans*. They can also form more complicated structure with parallelism and pooling steps involved as will be detailed in section 6.

4 Usage

Here, we describe how Substra can be used. First, the canonical use cases are detailed; second, the unitary operations allowed are listed; third, the interfaces to interact with Substra are described.

4.1 Use-cases

Three canonical use-cases for Substra are detailed: the *data / algorithm collaboration*, the *data consortium*, and the *Training / evaluation collaboration*. The first two use cases rely on the fact that efficient predictive models require lots of data to be trained. Thus, collaborations can be set up between several actors to increase the amount of data from which a model is trained.

Note that these use cases are compatible. Indeed, all the real world applications of Substra we are currently aware of borrow from the use-cases below.

4.1.1 Data / algorithm collaboration

When data controllers and model engineers belong to different organizations, effective collaboration can be a real challenge as data controllers may not be willing to transfer their sensitive data to potentially untrusted model engineers. Typically, data controllers host large amounts of sensitive data which can only be processed under strong confidentiality constraints. They have an incentive to limit the number of copies of their data, and are reluctant to provide access to the data itself. Model engineers design algorithms which often need large amounts of quality data to be used to build predictive models. Model engineers are always looking for more data to train their models and build new and better services. Thus, the collaboration would be beneficial to both but transfering the data is often to risky for the data controller. In this setup, Substra fosters collaboration by addressing and removing the need of data transfer.

For instance, in the precise use-case for which Substra was originally designed hospitals are the data controllers. Through their typical operations they collect countless sensitive and private data. They want to valorize this data in order to provide better patient-care or to foster medical research. Yet, they cannot share the data without constraints. In this use-case, model engineers are either academic researchers or companies specialized in medical AI which want to create and/or sell ML based services. These predictive services can be provided directly to the hospitals, or to some third party.

4.1.2 Data consortium

When competing entities separately collect very similar data, they may be interested in mutually improving the efficiency of their predictive model provided their data remains private. These organizations may want to collectiveley train an algorithm across their datasets and share the resulting aggregate model. Such an approach could improve the efficiency of the entire sector without favoring one actor over the other.

For instance, Substra can be used between several pharmaceutical companies which have almost identical processes to discover new drugs and have gathered very similar data over the years. Crucially, these companies are competitors and want to protect their data from each other. Yet, using Substra, they can collectively train a common predictive model without revealing their data. Thus, Substra helps them improve their ability to discover new drugs.

4.1.3 Training / evaluation collaboration

The practice of training predictive models is becoming widespread, but rigorous evaluation of performance is not always occuring. In most fields, the rise of ML is conditioned by the concrete proof that the ML models can generalize robustly and can be applied consistently on new data points. To measure the capacity of a ML model to generalize, a simple but efficient way is to evaluate it on a representative dataset it has never seen before: this is a test dataset.

Thus independent evaluators could gather representative, well-curated and non-biased datasets for testing the *Models* a posteriori. The Evaluators would design an *Objective* with a test dataset and open of leaderboard for the predictive models.

For instance in healthcare, one could imagine that regulatory bodies team up with strategic hospitals to register test cohorts so that all models in Substra can be benchmarked independently. For a given pathology, there could be test datasets "certified" by independent organisations to help evaluate *Models*. Then startups could design *Models* which are evaluated on independent data, leading to increased reproducibility.

4.2 Operations

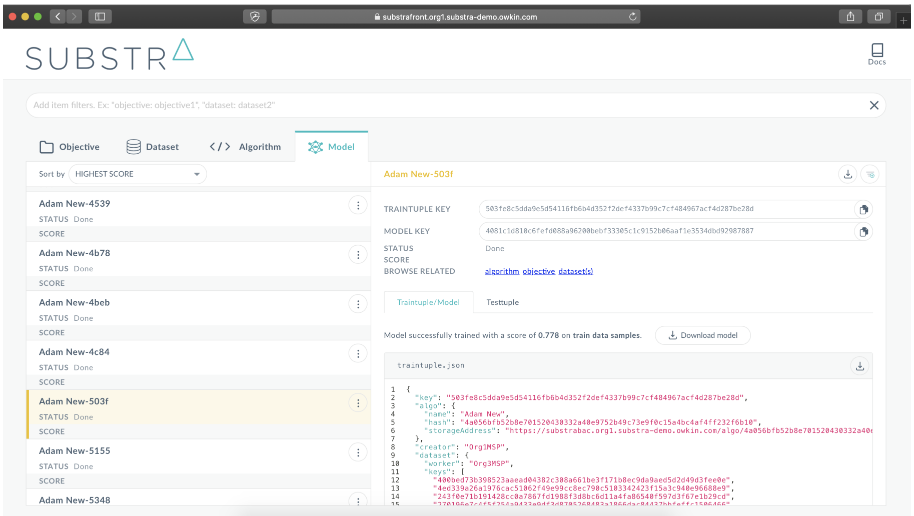
Substra makes it possible for a user to

- Create an *Asset* such as a *Dataset*, *Algorithm*, or *Objective*, via direct upload or regsitration from file.
- Change the *permission regime* of an *Asset*.
- Train a *Model* from available *Assets* by creating a *Traintuple*.
- Evaluate the performance of a *Model* on the test data of an *Objective*, or using a cross-validation approach on a *Dataset*, by creating a *Testtuple*.
- View and compare the performance of all *Models* in the form of a leaderboard (i.e. a list of models ordered by performance).
- Request a prediction from a *Model* on a new data point.

4.3 Interfaces

In order to perform these operations Substra comes with 3 types of interfaces: a web interface, a Command Line Interface (CLI), and a Python Software Development Kit (SDK).

- The **frontend** aims at providing traceability of all operations on Substra *Assets*. It can also be used to choose the desired *permission regimes* on *Assets*. As shown in figure 2, the Substra frontend displays lists of all *Assets* in specific tabs. A search bar can be used to filter *Assets*.



substrafront.org1.substra-demo.owkin.com
SUBSTRA
Docs
Add item filters. Ex: "objective: objective1", "dataset: dataset2"
Objective
Dataset
Algorithm
Model
Sort by
HIGHEST SCORE
Traintuple/Model
Testtuple
Download model

Adam New-4539
STATUS Done
SCORE

Adam New-4b78
STATUS Done
SCORE

Adam New-4beb
STATUS Done
SCORE

Adam New-4c84
STATUS Done
SCORE

Adam New-503f
STATUS Done
SCORE

Adam New-5155
STATUS Done
SCORE

Adam New-5348

Adam New-503f

TRAINTUPLE KEY 503fe8c5dda9e5d54116fb6b4d352f2def4337b99c7cf484967acf4d287be28d
MODEL KEY 4081c1d810c6fefd088a96200bebf33305c1c9152b06aaf1e3534dbd92987887
STATUS Done
SCORE
BROWSE RELATED algorithm objective dataset(s)

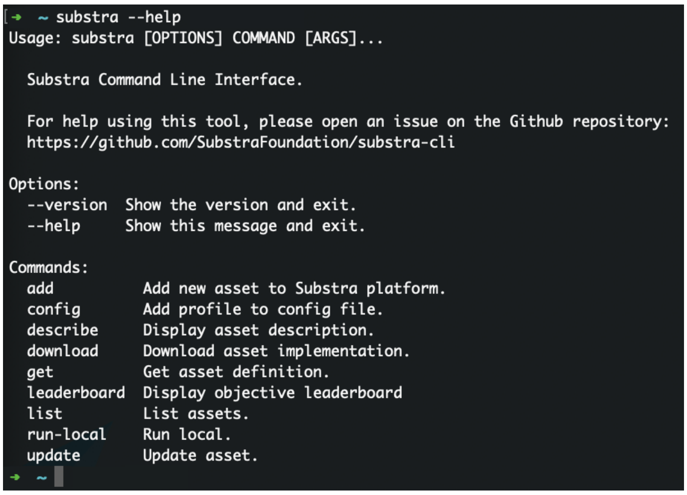
Model successfully trained with a score of **0.778** on **train data samples**.

traintuple.json

```
{
  "key": "503fe8c5dda9e5d54116fb6b4d352f2def4337b99c7cf484967acf4d287be28d",
  "algo": {
    "name": "Adam New",
    "hash": "4a056bfb52b8e701520430332a40e9752b49c73e9f0c15a4bc4af4ff232f6b10",
    "storageAddress": "https://substrabac.org1.substra-demo.owkin.com/algo/4a056bfb52b8e701520430332a40e
  },
  "creator": "Org1MSP",
  "dataset": {
    "worker": "Org3MSP",
    "keys": [
      "400bed73b398523aaead04382c308a661be3f171b8ec9da9aed5d2d49d3fee0e",
      "4ed339a26a1976cac51062f49e99cc8ec790c5103342423f15a3c940e96688e9",
      "243f0e71b191428cc0a7867fd1988f3d8bc6d11a4fa86540f597d3f67e1b29cd",
      "270196e7c4f5f254a9433e9df3d8705268483a1866dac84437bbfeffc1506466",
```

Figure 2: Screenshot of the Substra frontend. The *Model* page of the frontend shown here displays a leaderboard for a given *Objective*.

- The **CLI** makes it possible to add *Assets* to Substra and also to list registered *Assets*. Figure 3 shows the commands which can be executed with the CLI. Importantly, each *Asset* must be formatted in a proper way before being pushed to the platform.



```
[➜  ~ substra --help
Usage: substra [OPTIONS] COMMAND [ARGS]...

  Substra Command Line Interface.

  For help using this tool, please open an issue on the Github repository:
  https://github.com/SubstraFoundation/substra-cli

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  add          Add new asset to Substra platform.
  config       Add profile to config file.
  describe     Display asset description.
  download     Download asset implementation.
  get          Get asset definition.
  leaderboard  Display objective leaderboard
  list         List assets.
  run-local    Run local.
  update       Update asset.
➜  ~ 
```

Figure 3: Screenshot of the CLI help.

- The **Python SDK** provides the same functionalities as the CLI and offers the flexibility of a Python environment. As Python is a favored programming language of data scientists, it makes it easier for them to interact with Substra. In addition, the SDK makes it possible to integrate Substra in any Python-based application.

5 Architecture

Substra is a distributed software to orchestrate ML computation under tight privacy constraints. As opposed to the classical client/server architecture, Substra is fundamentally decentralized. It orchestrates the remote execution of ML tasks across several data centers. By design, the data never leave their original servers.

Inherently, Substra provides full traceability and control of data usage. At its core lies a decentralized and trustless consensus network which guarantees that all operations are orchestrated and written in an incorruptible ledger. No party can modify the ledger individually. The platform exclusively relies on the ledger to dictate its behavior, thus providing strong guarantees of traceability and reproducibility. The ledger in each node is identical. It records the history of all past, present, and schedulded operations on the network.

The various permission regimes individually governing each asset are also stored in the ledger as smart contracts. They have a regulating effect on the ledger in that the permission regimes filter the items added to the ledger. Every operation must meet all the permission constraints before it is added to the ledger and then subsequently executed by the platform. Since the smart contracts are self-enforceable, i.e. they do not rely on any third party to be executed, permission constraints are met by design.

Substra can be viewed as a secure API to run ML computations on third party data. In particular, it does not include automated design and execution of coherent training strategies over distributed datasets. Substra simply receives orders from users, checks their permission, and executes them.

5.1 Network architecture

Substra is decentralized: it runs on, and connects to, a set of machines in a private network. It is made of three parts: distributed nodes, a metadata network, and an asset network as shown in figure 4.

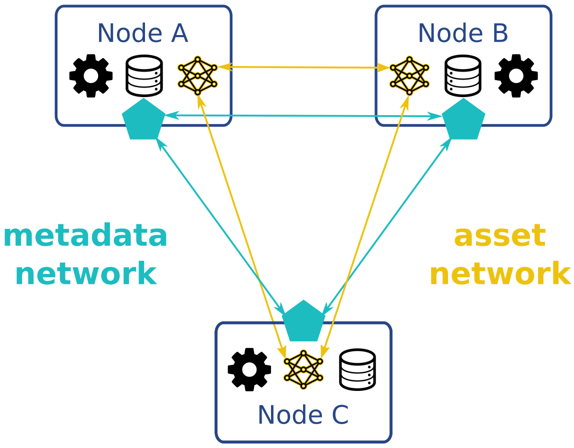
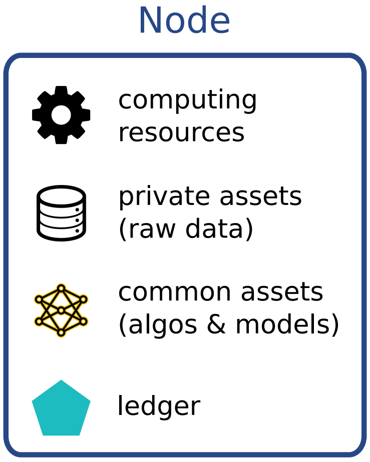
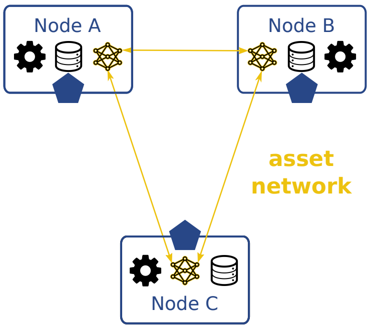


Figure 4: Global architecture of a Substra network.

- The Nodes are mades of four components
  - **Computing resources** can access private data and common assets in order to perform containerized computations. The computations are engaged only if specified in the ledger.
  - Storage of **private data** which never leave the node. This usually corresponds to raw sensitive data (e.g. medical data) which should remain private at all cost. The data are secured and used exclusively by the computing resources of the node.
  - Storage of **common assets** such as algorithms or trained models, which can be shared between nodes under permission constraints. They are used exclusively by the computing resources of the node.
  - A shared and immutable **ledger** which stores all the operations on the platform from computing tasks to data or models registration, and the complete permission settings associated to each *Asset*. Only non-sensitive metadata are stored in the ledger, as detailed in section 5.3. A library of smart contracts, called the chaincode, is used to read and write to the ledger.



- An asset network for selectively sharing common assets
  - The networks makes it possible to exchange authorized models and algorithms between nodes. More generally it is a channel to communicate large files with selected partner nodes.
  - Asset download is restricted for use by the node computation unit and by strict permission rules. All assets have explicit permission regimes which are checked systematically before manipulating them.
  - Private assets, e.g. sensitive raw data, are never shared on the asset network.



- A ledger network for sharing and updating the ledger
  - It is powered by a DLT framework: Hyperledger Fabric [5]. The ledger is consensually built and can not be corrupted. It is operated by the chaincode mentioned above.
  - The ledger of each node is updated frequently and consistently in order to register new *Assets*, set the *Asset* permissions, and append recent/requested computation tasks to the task history. As mentioned above, only non-sensitive metadata transit within this network.

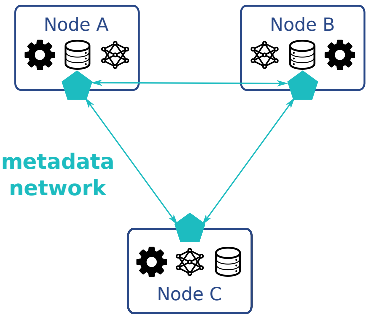


Figure 5 details the architecture of Substra, and interactions between its different components.

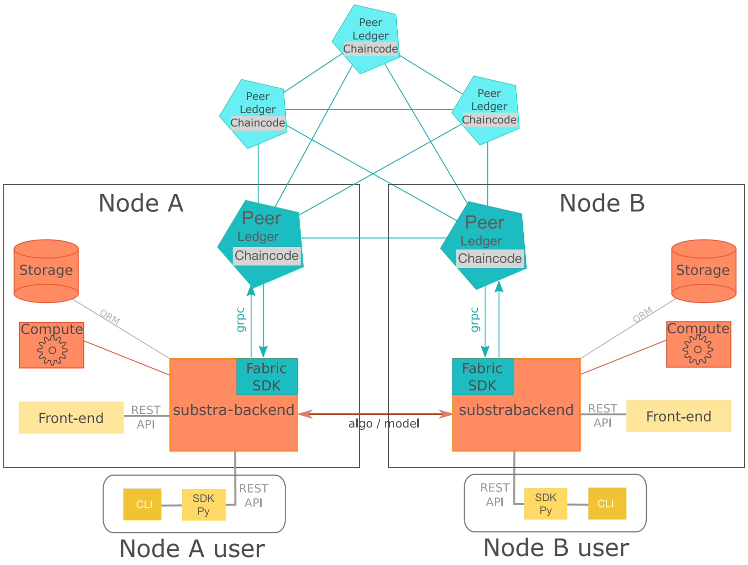


Figure 5: Overview of the architecture of Substra, illustrated with two nodes. Turquoise arrows indicate exchanges of non-sensitive metadata between the backend of a node and the ledger network. The red arrow indicate the exchange of an algorithm or model through the asset network.

5.2 Workflow

The workflow describes the typical sequence of steps that are performed by the platform in order to address user requests properly. It is out of the scope of this document to detail all of them, only the general and principal patterns are presented here.

5.2.1 Orchestration by the ledger of computations

The workflow of a Substra network is driven by two coupled and asynchronous circuits: ledger and computation operations.

- Ledger operations are performed through the DLT framework and are triggered by user inputs. They mainly consist in registering assets and specifying (possibly a sequence of) computations to be performed independently.
- Computation operations are automated and performed locally in the node with private and common assets. Computations are only triggered and authorized when defined in the ledger. Computation results are logged in the ledger.

Thus, if a node is randomly removed from the network then the other nodes can continue operating normally, except that they lose their ability to process the private data of the leaving node.

5.2.2 Example of workflow

To help understand the detailed workflow, a simplistic yet representative example between two nodes is detailed below. This corresponds to user A (owner of node A) being a data controller (e.g. a hospital) and user B (owner of node B) being an algorithm developer (e.g. an AI company). Both parties explicitly agree through the permission settings that user B can train her algorithms on user A's private data. Figure 6 illustrates the successive steps in the processing of user's B request.

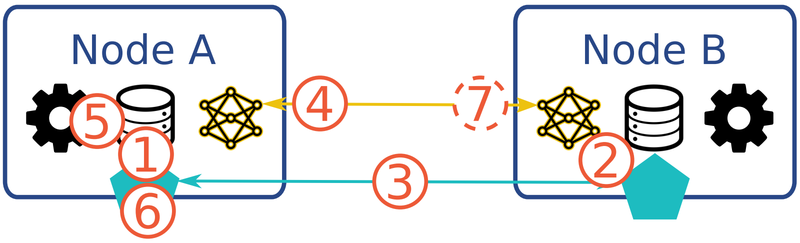


Figure 6: Substra's workflow between two example nodes. See description of steps in text.

1. User of node A registers private data to the platform. Associated non-sensitive metadata are submitted to the ledger (see details in section 5.3). The ledger update is automatically broadcasted and accepted by the other node.

2. User of node B registers a common/shared algorithm to the platform. Associated non-sensitive metadata are submitted to the ledger. The ledger update is automatically broadcasted and accepted by the other node.

3. User of node B requests a computation of his algorithm on user A's private data. A new computation task is specified in the ledger. It is then broadcasted and accepted by the other node only if it meets the permission settings of node A's private data, which are found in the ledger.

4. Node A observes that there is a new task to be processed in the ledger and automatically downloads the common algorithm from node B (which authenticates node A against the ledger) and stores it.

5. Node A securely performs the computation and applies user B's algorithm on its own private data.

6. Non sensitive metadata summarizing the computation execution and resulting performance are written by Node A in the ledger which synchronizes with other nodes immediately.

7. (Optional) Outputs of the computations, such as trained models (but never private data), can be sent back to node B depending on chosen permissions.

This simple workflow can be chained and composed to perform arbitrarily complicated computations at the scale of the network, as detailed in section 6.

The ledger gathers all user inputs and the computation units listen to ledger updates in order to trigger and perform computations on their private data.

5.3 Information stored in the ledger

As mentioned previously, the ledger stores only non-sensitive metadata, required for the orchestration and for the traceability of the training of machine learning models on distributed data.

Table 5.3 summarizes elements stored in the ledger.

Object	**Attributes stored in the ledger**
Objective	- Name of the *Objective* - Storage address and hash of its description - Name, storage address and hash of its metrics - Owner (node who defined the objective) - Test datasets (list of data keys for the test split, and their associated dataset) - Permissions
Dataset	- Name of the *Dataset* - Storage address and hash of its data opener type of data in the dataset (tabular, image, ...) - Storage address and hash of its description file - Owner - Associated *Objective* key - Permissions
Data	- Hash of data stored in local storage - List of keys of associated *Datasets* - A boolean indicating if data is dedicated to testing
Algorithm	- Name of the *Algorithm* - Storage address and hash of the algorithm files - Owner - Associated *Objective* key - Storage address and hash of the description of the algorithm - Permissions
Traintuple	- Associated *Objective* key (for its metrics) - Associated *Algorithm* key - List of input models (list of traintuple keys, hashes, addresses) - Output *Model* (hash, address) - List of training data and the node where they are stored - Status of the task: waiting, todo, doing, done, failed - Log - Optional arguments necessary for complex ML orchestration (a rank and a tag) - Permissions - Creator (node who defined the traintuple)
Testtuple	- Associated *Objective* key (for the metrics) - Associated *Algorithm* key - Model to evaluate (hash, address) - List of testing data and the node where they are stored - Status of the task: waiting, todo, done, failed - Log - Optional arguments necessary for complex ML orchestration (e.g. a tag regrouping several ML tasks) - Permissions - Creator (node who defined the testtuple)

6 ML orchestration

To launch Substra on large amounts of distributed data, users must create sequences of tasks which are executed by Substra. These are called *compute plans*. They prescribe unambiguously the organization and ordering of computations for training algorithms on datasets. They also make it possible to evaluate a model against several test datasets. Substra does not involve automated generation of *Compute plans*. It simply executes the user's orders in the form of *Compute plans*.

6.1 Chaining training and averaging

The first building block of *compute plans* is the **training step**, specified by a *traintuple*. It defines a unitary training task which updates a *Model* as the result of the training of an *Algorithm* on a given *Dataset*. In the classical setup *traintuples* take a single model as input and provide another model as output.

The second building block of *compute plans* is the **averaging step**. It takes several *Models* as inputs and outputs a single "averaged" *Model*, either by averaging the prediction of the input models [15], or by averaging the weights of the inputs models [22]. Depending on the type of model used and the particular FL strategy, more complex operations might be needed.

A *compute plan* is a set of training and averaging steps, whose in and out models are chained in a specific and possibly complicated pattern. Basically, chaining can be done in a sequential or parallel way as illustrated in figure 7.

Sequential compute plans only use training steps; there is no averaging involved. In the simplest case, the *Algorithm* is fixed and a *Model* is successively trained on multiple remote *Datasets*. Note that changing the order of the datasets is likely to change the end model.

Parallel compute plans correspond to a sucession of training and averaging steps. First, several models are trained from the same initial model using different training datasets. Second, the models are aggregated in a single output model.

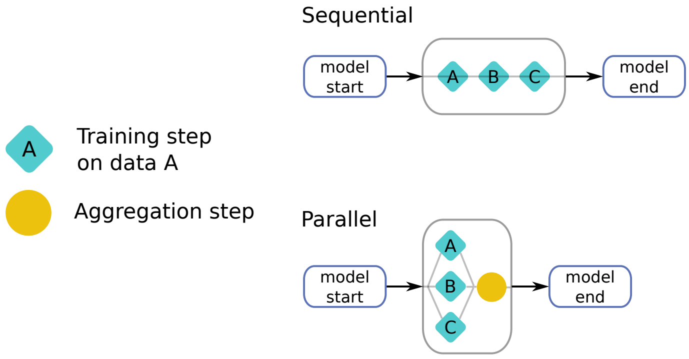


Figure 7: (left) Elements of *compute plans* (see text for details). (right) Basic *compute plan* samples: (i) sequential training consists in a sequence of training on different datasets, (ii) parallel learning is made by training several models independently but with the same initial model, then an aggregating step merge the several models in a single output.

These two basic patterns can be chained and composed to create arbitrarily complex *compute plans* as illustrated in figure 8. The first *compute plan* shown in figure 8 corresponds to the standard pattern in federated learning [22].

6.2 Evaluation

Evaluation is an important part of Substra. It is split in validation datasets for hyper-parameter tuning and test datasets for evaluation on new data.

- **Validation** is made flexible and entirely parametrable by users who can perform any kind of cross validation scheme on any subpart of the train datasets.
- **Test** datasets however are sanctuarized and can never be used for training. Test performance evaluation is constrained to a rigorous methodology.

Each node in the network may define a subpart of its data as an immutable test dataset. The evaluation against these datasets is defined in the form of an *Objective* and can be requested by any user with the appropriate processing rights. Figure 9 illustrates a typical evaluation pattern of the performance of a model before and after training. Notably, the evaluation in Substra can be done on each user dataset. Thus, there may be different performance levels of a single model depending on the node/user evaluating it.

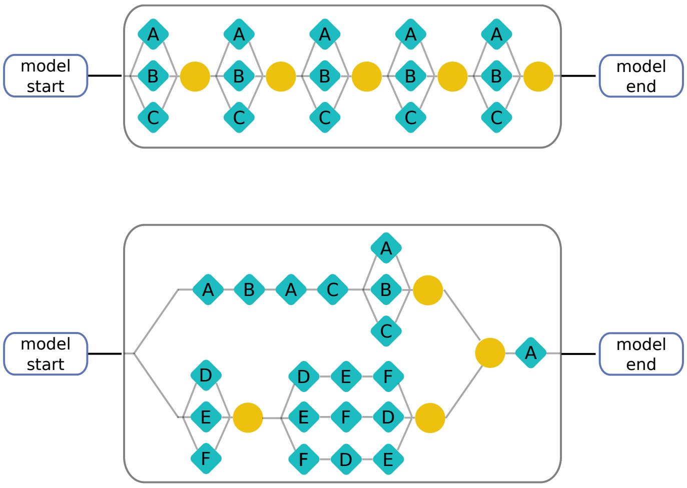


Figure 8: Sample *compute plans*. (top) Standard *compute plan* with regular succession of parralel and averaging steps. (bottom) Excessively complicated sample *compute plan* to illustrate the compostion of basic patterns.

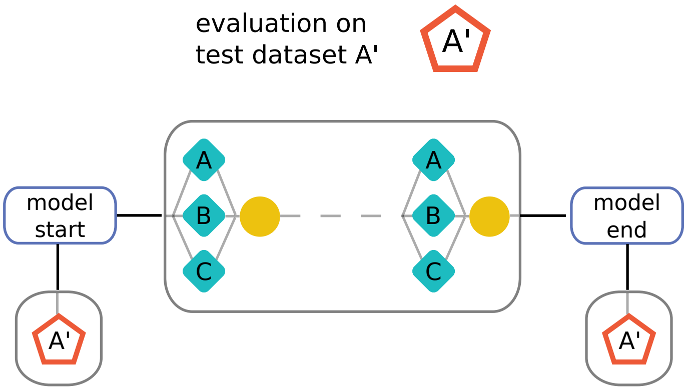


Figure 9: Sample *compute plan* involving evaluation steps. Here the owner of data A and A' is training an algorithm on other remote datasets. Although it involves other data, the evaluation is only made on A'.

Cross-validation is a standard methodology for evaluating model performance. For a k-fold cross-validation, k pairs of training and test datasets are derived from the original dataset, and the performance is computed as the average of the performance on the k test datasets. In Substra cross-validation is formalized as a specific *Objective* with a training dataset, but without test dataset.

6.3 Model composition

Large ML models, in particular deep learning models, are often the composition of several sub models which can be trained independently. Substra supports models defined as combinations of others. Of course, the complexity of the resulting algorithm has to be handled by the user; but Substra has been specifically designed to make it easy to handle training tasks over combined models.

Transfer learning on neural networks often implies modifying an existing network by adding or removing some layers and fine tuning some connections in the network. For instance, when considering a task of classification over images, it is common to download the weights of a pre-trained neural network designed for discriminating other classes (such as ResNet [19]) and recycle the lower layers of the deep network as shown in figure 10. This corresponds to what is called a warm start. It makes it possible to reuse the weights trained for a certain *Objective* for another one. This is the canonical example of sequential orchestration with model composition. In fact, figure 10 corresponds to a simple sequential *compute plan*.

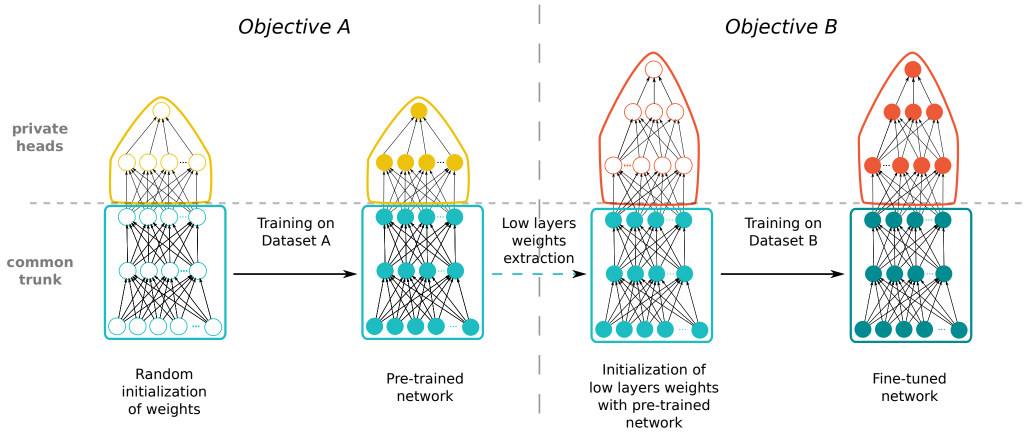


Figure 10: Classical multi-task learning approach called warm restart. First, a neural network is trained on *Dataset* A and evaluated against *Objective* A. The initialization of the weights is usually random. In order to transfer the knowledge from the pre-trained *Model* to another task, one can extract its lower layers and consider them as the initial weights of a second network built for *Objective* B, which is to be fine-tuned on *Dataset* B. In this case the inputs are considered the same across *Objectives*.

This means several users can decide to share exclusively the lower layers of a network. This would correspond to changing the *process permission* of the sub-network. This common subpart of the *Model* is called a *Trunk*. Each node keeps the upper layers of its *Model* private. They constitute the private *Heads*. Figure 10 illustrates a situation where two partners want to train together a common trunk model, but do not want to share their data and not even the definition of their own *Objective*. It is a situation with increased privacy where partners do not know what the other is computing. Nonetheless, in using the backpropagation training algorithm not only through head layers, but also through trunk layers for each partner, the trunk model can benefit from the information of all partners without revealing private information.

Substra is also designed to tackle parallel orchestration combined with model composition. A standard compute plan associated with such a combined model is illustrated in figure 11.

Note that a quasi identical approach can be applied for domain adaptation [24], which corresponds to multiple users having an identical goal (and thus a single *Objective*), but with data from

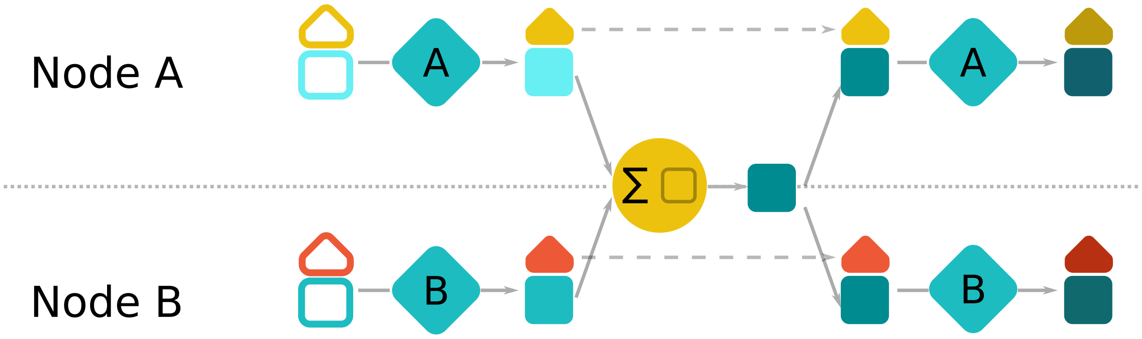


Figure 11: Federated multitask learning. The trunk model is trained collectively but the private head models are never shared between partners.

different sources and with slightly different format or distribution. In this case, the modularity of the network occurs at the lower layers of the neural network.

7 Risk analysis

This section proposes a high-level risk analysis focused on the Confidentiality, Integrity, and Availability [1] of the different *Assets* of Substra. An exhaustive risk analysis is out of the scope of this whitepaper, and we focus on the risks which we have estimated as the most relevant. In particular, we focus on the risks due to an attacker being part of the network, i.e. the attacker is assumed to control entirely its own node and is willing to attack the *Assets* of others. For simplicity, we do not address the risks due to attackers from within the node or from outside the network. These kinds of attacks are relevant and should be mitigated, but are not specific to Substra, thus they are not addressed here.

An important feature of Substra is trustless traceability. Any risk is at least partly mitigated by transparency mechanisms. It is not possible to launch operations which are not stored in the ledger with Substra. Thus, any attack triggering specific computations will be recorded in the ledger for possible future inspection.

7.1 Confidentiality

Substra is deliberately developed to provide high confidentiality of *Datasets*. In many situations confidentiality is synonymous to privacy, which is terminology specific to personal data. At the core of Substra design is the principle of never moving *Datasets* between nodes. Only the *Algorithms* and *Models* are exchanged between nodes. Thus, compared to a classical centralized architecture, using Substra decreases the confidentiality risk on *Datasets* at the expense of an increased confidentiality risk on *Algorithms* and *Models*. Substra implies transfer of risk from *Dataset* to *Algorithms* and *Models*.

The confidentiality attacks can be organized in 3 groups: *Dataset* theft, *Algorithm* / *Model* theft and metadata leak. They are identified in the table below:

Risk	**Risk description**	**Built-in Substra mitigation**	**Additional mitigations needed**
Data theft	Attacker downloads raw data.	Data remain within each node infrastructure.	
	Attacker designs malevolent *Algorithm* to extract data (possibly in the *Model* weights).	*Model*'s permission regime can forbid model download. Securing the compute worker (e.g. no access to network during computations).	*Algorithm* certification by third party.
Data inference	Attacker infer properties about the data from the trained models.	Model access can be restricted.	Certification for designing non-identifying models. Contractualisation between partners.
Algorithm / Model theft	Attacker steals the algorithm or model during a training step.	Securing the compute worker (e.g. model never stored on hard drive). Permission regime selects who can process model.	Trusted Execution Environments (TEE). Contractualisation between partners.
Metadata leak	The common ledger containing the platform metadata is leaked outside the network.	Metadata are anonymous, the ledger only contains hashes of assets.	

Overall, the risk of a *Dataset* confidentiality breach is largely mitigated by the Substra computation architecture. The main residual risk lies in malevolent *Algorithms* which could leak *Datasets* out of the node, for instance, in writing the data themselves in the model weights. It can be strongly mitigated by requiring the *Models* permission regimes to exclude download access to the corresponding *Algorithm* designer. Independent audit and certification of the *Algorithm* before deployment could be a pragmatic alternative. A more limited residual risk consists in infering properties about the individual data from the trained model itself. This is actually unlikely if the training procedure is well designed and the model diffusion limited.

The residual risk on *Algorithms* and *Models* confidentiality is not negligible. On its own, Substra can only make it difficult for a node to steal a *Model* which is being trained locally. Since the program runs on the machine of the user it cannot be theoretically bulletproof. A future idea for a solution would be to rely on the growth of Trusted Execution Environments [26, 21], but the lack of required GPU support for current TEE make this possibility speculative. In practice, it is necessary to consider contractual interactions between the network partners to cover the residual risk on *Algorithms* and *Models.* Today, Substra is not likely to be deployed in an open network where everyone could participate as in a public blockchain.

Confidentiality of *Datasets* goes beyond the simple access to the raw data; privacy of individuals who are part of the *Dataset* is to be guaranteed. The design of Substra was strongly influenced by the GDPR and takes the question of privacy as a first principle. More precisely, an important risk to address is the leaking of high level personal information without accessing the *Datasets* themselves.

There are several guidelines or good practices that we strongly recommend to adopt when deploying Substra over personal data. Their details are out of the scope of the whitepaper.

- Always **pseudonymize** data before registering them as *Datasets* in Substra. Anonymization procedures [8] are even better, but not always relevant for certain use cases (e.g. Healthcare where re-identifying patients is key to provide personalized care).

- Design **non-identifying** *Models*, so that they can be exchanged without exposing personal information. *Models* are statistical objects which do not need to contain identifying information. Actually, it is often the case that models that leak information about individual data points are ill-designed for this setting; for instance in the classical problem of overfitting ([29]). Requiring training *Algorithms* to be properly regularized is paramount. Similarly, an efficient mitigation consists in restricting training steps to be performed on large groups of individuals so as to dilute the personal information in the aggregate. Trusted audits and certification of *Algorithms* are necessary for the use cases requiring the highest degree of privacy.

In other words, Substra does not solve the problem of privacy on its own; a number of privacy-enhancing protocols must be enforced to guarantee the highest level of privacy.

7.2 Integrity

Substra provides high integrity guarantees, mainly because of its architecture rooted in an unfalsifiable distributed ledger logging all ML operations. All *Assets* are registered and referenced through a unique identifier. The identifier is in fact the hash of the *Asset*, which makes it possible to guarantee the *Assets* have not changed when reproducing a sequential training procedure. Integrity of *Assets* can therefore be checked at all times.

Integrity applies also on the results, i.e. predictions on new data. It is crucial to make sure that the predictions cannot be biased by an attacker. Beyond standard attacks on a node which are not covered here, there is a particular kind of attack fundamentally linked to the specifics of Substra: participating in a collaborative training (with *Datasets* or *Algorithms*) in order to bias the prediction of the output *Model*. A possible mitigation of this risk is to test the predictive performance on specific test *Objectives* equipped with controlled and independent test *Datasets*. It is indeed likely that methods devoted to bias prediction lead to poorer performance on test datasets. There is a clear security incentive to consider and favor the best *Model* for each *Objective*. However, a residual risk lies in the fact that large predictive models could contain stolen data while keeping good performance.

Risk	**Risk description**	**Built-in Substra mitigation**	**Additional mitigations needed**
Results integrity	Attacker changes the performance metadata.	DLT prevents from changing reports made by others.	
Results and *Assets* integrity	Attacker modifies a model to alter prediction quality (for instance by training on biased datasets).	Full traceability of operations and storing of intermediate models to be able to revert a bad training step.	Independent test sets to validate performance evolution

7.3 Availability

Availability attacks are not critical to Substra functioning. Being a distributed, asynchronous network, Substra has a low-latency. This is not a significant problem since Substra is focused on training *Models* which does not involve immediate user interaction. It is also resilient to attacks on single nodes due to its decentralized architecture: an availability attack on a partner only blocks the *Assets* exclusively owned by this partner.

To decrease the risk of overloading a node, the ledger or the network with too many training or prediction tasks, permission regimes can be set up to create a white list of users which can request computations. The permissions are applied a priori to prevent illegitimate training tasks to be added in the ledger and synchronized over the network.

Risk	**Risk description**	**Built-in Substra mitigation**	**Additional mitigations needed**
Service availability	Attacker overloads a node with training or prediction requests.	Other nodes still function. Permission regime authorizes only specific nodes to launch computations on one's node.	

8 Perspectives

Substra's ambition is to become the standard framework for performing collaborative ML over distributed datasets. It is designed to be modular and open. There are countless ways to improve the framework or integrate it into existing software solutions.

Integration with similar projects

There are several technologies similar to Substra which are emerging today; we believe the future of Substra is conditioned by its ability to interface nicely with other software in particular in the open source community. Collaborative initiatives between projects are likely to drive the rise of a "responsible and trustworthy data science" ecosystem that we pledge for.

To name only a few similar projects:

- OpenMined [4] is building PyTorch libraries for privacy-preserving deep learning with deep training content [10].
- Dropout Labs [2] is building TensorFlow libraries for ML on encrypted data.
- Google is designing a library/SDK for TensorFlow to add federated learning features for smart phones [9].
- Ocean Protocol [7] and Oasis Labs [6] are building decentralized networks enabling the development of privacy-preserving data-based applications.

Secure aggregation of model updates

In parallel compute plans, there is a central aggregator which takes many model updates from different nodes as inputs and outputs a single averaged model. This is a typical pattern in Federated Learning [22]. It is associated with a significant risk at the central point since the aggregator has access to all model updates from the nodes, from which it could infer sensitive information about the nodes' data. A classical approach to mitigate this risk is to use a mechanism of secure aggretion where the model updates are individually obfuscated by the aggregator can still output an accurate average [13]. This is an interesting perspective for Substra for some use-cases.

Non assignability of metadata

Substra could benefit from strong anonymity features regarding who owns which *Asset*. For now, assigning the ownership of an *Asset* to a node is accessible to competent and technical attackers. However, the DLT-based architecture of Substra gives us reason to assume that providing anonymity is technically possible. Substra is still a consortium based technology; thus it will only provide anonymity among the members of an explicit consortium. But "hiding among the trees" can be relevant to numerous use cases.

Ownerless Assets

Substra could open the possibility of having ownerless *Assets*. By combining a Multi Party Computation approach (such as Shamir secret sharing) with the ledger of Substra, one could imagine encrypting some *Assets* and split the decryption key among nodes. Later when a *traintuple* involves this *Asset*, a node could gather all the key parts from other nodes to decypher the *Asset* which could be erased from this node after computation. This feature could open brand new use cases where extra privacy or decentralized control of *Assets* is required. This raises difficult technical and security challenges, for instance, to keep the availibility of service sufficiently high for practical applications.

Non-identifying Models

As shown in the risk analysis in section 7, the usage of Substra needs to be considered together with the definition and adoption of sound guidelines for *Algorithm* design in order to reach the highest level of privacy preservation. The guidelines should guarantee that *Models* are non-identifying. This means that the precise information about data samples in the train *Datasets* cannot be retrieved from a *Model*. For instance, these guidelines will surely address the problem of overfitting for *Model* design. This problem is not only bad for generalization of *Models*, but it also leaks much more information about the overfitted *Dataset*. Importantly, these guidelines will have to be promoted and checked by trustful regulatory entities.

Partner ecosystem

On an ecosystemic aspect, the growth of Substra into a widespread, production-grade network is likely to involve different roles among the partners within a Substra network. Inevitably, the growth of Substra will be bounded by the capacity of the current actors of our socio-economic ecosystem to structure themselves to use the technology to their specific advantage. A first category of actors gathers *Asset* controllers. Some of them will specialize in *Datasets* collections and management (e.g. Hospitals), whereas some others will specialize in *Algorithm* creation and management (e.g. AI startups). Both types of actors will inevitably have to deal with *Model* management and permission setting. A second category of actors gathers regulators and evaluators of the *Models* performance. They will deal with the design and management of *Objectives* and in particular the associated test *Datasets*. Note that it is crucial to have independent actors providing standardized benchmarks of *Models*.

Token-based economic ecosystem

Finally, an interesting perspective would be to leverage the built-in traceability within Substra in order to create a token-based economic system for collaborative Machine Learning. This would involve attributing a value to the Substra *Asset* which could be aligned with the performance improvement brought by each *Asset* on predefined *Objectives*. For instance, one could compute a contribution score for each *Datasets* by evaluating the percentage of improvement observed after training on it. Although it is beyond the scope of the current version of Substra, this economic system could be a real driver for the growth of Substra and collaborative, privacy-preserving Machine Learning.

Acknowledgement

This work is funded by Owkin and Bpifrance as part of the ”Healthchain” project, which resulted from the ”Digital Investments Program for the major challenges of the future” RFP.

We are grateful to the open source community for all the great libraries at the basis of our work. Standing on the shoulders of giants as they say.

Special thanks to Eric Boniface, Kelvin Moutet, Guillaume Cisco, Eric Tramel, Samuel Lesufleur, Clément Gautier, Romain Bey, Romain Goussault, Inal Djafar, Clément Mayer, and Talia Litteras for helping improve the manuscript.

We thank the developers of Substra, the members and staff of Substra Foundation and the employees of Owkin for all the useful discussions, experiments, and contributions that led to this document and the associated source code.

References

[1] Confidentiality, integrity, and availability. `https://developer.mozilla.org/en-US/docs/Web/Security/Information_Security_Basics/Confidentiality,_Integrity,_and_Availability`. Accessed: October 2019.

[2] Dropout labs tf encrypted libraryw. https://github.com/tf-encrypted/tf-encrypted. Accessed: June 2019.

[3] General data protection regulation. https://eur-lex.europa.eu/eli/reg/2016/679/oj. Accessed: 2019-06-11.

[4] A generic framework for privacy preserving deep learning. https://arxiv.org/pdf/1811.04017.pdf. Accessed: June 2019.

[5] Hyperledger fabric. https://www.hyperledger.org/projects/fabric. Accessed: 2019-06-11.

[6] Oasis labs platform overview. http://docs.oasiscloud.io/en/latest/overview/. Accessed: June 2019.

[7] Ocean protocol technical whitepaper. https://oceanprotocol.com/tech-whitepaper.pdf. Accessed: June 2019.

[8] Opinion 05/2014 on anonymisation techniques. https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp216_en.pdf. Accessed: 2019-06-11.

[9] Tensor flow federated. https://www.tensorflow.org/federated. Accessed: October 2019.

[10] Udacity 'secure and private ai' course. https://eu.udacity.com/course/secure-and-private-ai--ud185. Accessed: June 2019.

[11] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[12] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[13] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[14] Nick Bostrom and Eliezer Yudkowsky. The ethics of artificial intelligence. *The Cambridge handbook of artificial intelligence*, 316:334, 2014.

[15] Gerda Claeskens, Nils Lid Hjort, et al. Model selection and model averaging. *Cambridge Books*, 2008.

[16] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015.

[17] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[18] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. 2009.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Brian Hie, Hyunghoon Cho, and Bonnie Berger. Realizing private and practical pharmacological collaboration. *Science*, 362(6412):347–350, 2018.

[21] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanovic. Keystone: An open framework for architecting tees, 2019.

[22] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[23] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[24] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[25] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

[26] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 2015.

[27] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.

[28] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[29] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 268–282. IEEE, 2018.