A Privacy-Preserving Distributed Analytics Platform for Health Care Data

Sascha Welten[1] Yongli Mou[1] Laurenz Neumann[1] Mehrshad Jaberansary[1] Yeliz Yediel Ucer[2] Toralf Kirsten[3] Stefan Decker[1,2] Oya Beyan[2,4]

[1] Chair of Computer Science 5, RWTH Aachen University, Aachen, Germany
[2] Department of Data Science and Artificial Intelligence, Fraunhofer FIT, Sankt Augustin, Germany
[3] Department of Medical Data Science, University Medical Center Leipzig, Leipzig, Germany
[4] Institute for Medical Informatics, Faculty of Medicine, University Hospital Cologne, University of Cologne, Cologne, Germany

Address for correspondence Sascha Welten, MSc, Chair of Computer Science 5, RWTH Aachen University, Ahornstr. 55, 52056 Aachen, Germany (e-mail: welten@dbis.rwth-aachen.de).

Methods Inf Med 2022;61:e1–e11.

Abstract

Background In recent years, data-driven medicine has gained increasing importance in terms of diagnosis, treatment, and research due to the exponential growth of health care data. However, data protection regulations prohibit data centralisation for analysis purposes because of potential privacy risks like the accidental disclosure of data to third parties. Therefore, alternative data usage policies, which comply with present privacy guidelines, are of particular interest.

Objective We aim to enable analyses on sensitive patient data by simultaneously complying with local data protection regulations using an approach called the Personal Health Train (PHT), which is a paradigm that utilises distributed analytics (DA) methods. The main principle of the PHT is that the analytical task is brought to the data provider and the data instances remain in their original location.

Methods In this work, we present our implementation of the PHT paradigm, which preserves the sovereignty and autonomy of the data providers and operates with a limited number of communication channels. We further conduct a DA use case on data stored in three different and distributed data providers.

Results We show that our infrastructure enables the training of data models based on distributed data sources.

Conclusion Our work presents the capabilities of DA infrastructures in the health care sector, which lower the regulatory obstacles of sharing patient data. We further demonstrate its ability to fuel medical science by making distributed data sets available for scientists or health care practitioners.

Keywords
- distributed analytics
- Personal Health Train
- FAIR

received
March 30, 2021
accepted after revision
September 22, 2021
published online
January 17, 2022

DOI https://doi.org/10.1055/s-0041-1740564.
ISSN 0026-1270.

Georg Thieme Verlag KG, Rüdigerstraße 14, 70469 Stuttgart, Germany

Introduction

In health care environments, such as hospitals or medical centres, a large amount of data is collected describing symptoms, diagnoses, and various aspects of the patient's cure process. The recorded data is usually reused for reviewing and comparing patient's state at the time when the patient visits the medical center again. In few cases, selected data—sometimes data of specific investigations—is shared for continued patient care process, for example, when the patient moves to another hospital. Apart from this data sharing, health care data is one fundamental source for medical research. This includes building cohorts for upcoming or running clinical studies as well as investigating in testing medical hypotheses and determining patients in emergent situations, such as virus outbreaks. The study results often depend on the number of available patient data. Typically, the more the data is available for the intended analysis or the scientific hypotheses, the more stable the results are. However, the reuse of patient data for medical research is often limited to data sets available at a single medical centre. The most immanent reasons why medical data is not heavily shared for research across institutional borders rely on ethical, legal, and privacy aspects and rules. Such rules are typically guarded by national and international laws, such as General Data Protection Regulation (GDPR: gdpr-info.eu) in Europe, Health Insurance Portability and Accountability Act (HIPAA:www.hhs.gov/hipaa) in the U.S., or the Data Protection Act (DPA:www.gov.uk/data-protection) in the U.K. limiting sharing sensitive data. These limitations tremendously affect medical research directions, in which a data set is not sufficiently available at each single hospital. Due to these limitations, solutions for distributed analytics (DA) have fuelled the progress of privacy-preserving data analysis by bringing the algorithm to the data. Consequently, by design, sensitive data never leaves its origin and the data owner keeps the sovereignty over the data. Therefore, this paradigm shift poses a first step towards compliance with the above-mentioned regulations.

In this work, we present our DA infrastructure, which has been built upon the established so-called Personal Health Train (PHT) paradigm. We further present novel, data provider-centric, and privacy-enhancing features enabling transparency of the activities within the DA ecosystems and privacy preservation of possibly sensitive analysis results. Lastly, we briefly compare our infrastructure with other DA implementations.

The remainder is structured as follows. The "Background" section gives an overview of the DA landscape and presents similar approaches and solutions. In the "Methods" section, we give insights into our design decisions and several integrated key features. The "Analytical Tasks" section covers a data use case demonstrating the capabilities of our implementation, and in the "Discussion" section, we contextualise our solution in the DA landscape and discuss the legal aspects concerning data privacy. Finally, the last section concludes this article and gives an outlook for future work.

Background

In recent years, several emerging technologies and approaches have been proposed to enable privacy-preserving (scientific) usage of sensitive data in the scope of DA and decentralised data.[1–6] In general, two prominent DA paradigms can be derived. One paradigm is the parallel approach. Architectures following this pattern send multiple replicas of the analysis including queries, algorithms, or statistical models to the data providers and have a protocol to train a data model—so-called Federated Learning (FL)—or an aggregation component, which merges the (query) results of each client.[3,5] In contrast to this parallel approach, there exists a nonparallel paradigm.[1,4] Instead of a central aggregation server, the intermediate results are transmitted directly or via a handler unit from data provider to data provider. The results are incrementally updated and returned to the requester after a predefined number of data provider visitations. Some recent works defined this as institutional incremental learning or weight transfer.[1,7]

Several infrastructures and platforms are based on these (abstract) principles. One concept following the latter paradigm is the above-mentioned PHT, which has been applied to several use cases by the scientific community.[8–12] From a top-level perspective, the PHT uses containerisation technologies to encapsulate the statistical algorithm, which is transmitted from institution to institution. These containers are executed on-site without the need to preinstall any dependency, which enables flexibility with respect to the used programming language and the data storage technology.[8] Another DA-enabling technology is DataShield.[13] DataShield is a distributed infrastructure, which primarily follows the federated (client-server) analytics approach, intended to conduct nondisclosive analysis of biomedical, health care, and social science data.[13–15] Instead of containerising the algorithmic code, DataShield relies on REpresentational State Transfer (REST) interfaces establishing the connection between the DataShield client and a DataShield server (OPAL server: opaldoc.obiba.org) installed at each institute, which receives the analysis command and executes it. For the execution of the analysis, DataShield uses the programming language R as the framework for the statistical analysis.

An additional concept, which can be interpreted as part of DA, is secure multiparty computation (SMPC). SMPC involves a cryptographic protocol to calculate a result set, where the data of each participating party remains concealed. A simple protocol might be noise induction (additive secret sharing), which makes a (malicious) derivation of the actual result set ambiguous.[16–18] However, more complex protocols applying homomorphic encryption techniques are also feasible.[19] While multiple SMPC frameworks have been developed in the past, most approaches are impeded by the connection and network setup of the participating entities due to the computational overhead induced by the chosen SMPC protocol. Especially, if the level of computational complexity increases (e.g., additional parties or more complex tasks), the network bandwidth could pose a significant bottleneck

yielding inefficient runtimes. Despite these community- and research-driven solutions, commercial DA solutions have attracted attention during the past years. FL for health care practitioners has been powered by the NVIDIA Clara FL software development kit (SDK) (developer.nvidia.com/blog/federated-learning-clara/). For example, the initiative EXAM, involving 20 individual hospitals, used the Clara framework to train a model, which predicts oxygen needs for COVID-19 patients (https://blogs.nvidia.com/blog/2020/10/05/federated-learning-covid-oxygen-needs/). Further, Tensorflow Federated (TFF) has been proposed by Google (www.tensorflow.org/federated), which has been developed to facilitate collaboratively train a shared model on separated locations.

However, these commercial products restrict the users to the provided software and hardware, for example, both Clara FL and TFF are based on Tensorflow/Python and Clara FL is built upon the so-called NVIDIA EGX edge computing platform (www.nvidia.com/en-us/data-center/products/egx/). These circumstances pose challenges with respect to the applicability of such platforms since, on the one hand, the data providers have very strict and varying regulations concerning third-party software executed on their site, which complicates the data provision even more. On the other side, these commercial solutions are limited by the provided frameworks or SDKs. This makes these solutions inflexible with respect to other popular programming languages like R (programming language dependency).

Nevertheless, while R is a well-established tool for statistical analysis, it also lacks functionality with respect to the training of complex deep learning models with DataShield since the R toolbox is limited to selected functions and methods.[13,15] Moreover, the DataShield packages are deployed to an OPAL server, which restricts the analysis to a predefined data source executing the R commands (www.datashield.org/help/get-started).

In the next sections, we present our DA infrastructure, which fuses the programming language- and data source-independence by using containerisation technologies. We exemplary show the current implementations in the context of the above-mentioned PHT approach and we present novel features, which constitute valuable options to eliminate common shortcomings like nontransparency of DA ecosystems and customisability.[20]

Methods

This section covers the architectural design of our implemented DA platform. We first provide an overview of all involved components before we present the features, design principles, and technologies of each component in detail. Please refer to the **Supplementary Materials**, available in the online version only.

Overview

The PHT originates from an analogy from the real world. The infrastructure reminds of a railway system including trains, stations, and train depots. The train uses the network to visit different stations to transport, for example, several goods.

Adapting this concept to the PHT ecosystem, we can draw the following similarities. The Train encapsulates an analytical task, which is represented by the good in the analogy. The data provider takes over the role of a reachable Station, which can be accessed by the Train. Further, the Station executes the task, which processes the available data. The depot is represented by our Central Service (CS) including procedures for Train orchestration, operational logic, business logic, and data management.

Furthermore, we pay attention to the following design aspects. Every component of our architecture is containerised using the Docker (www.docker.com) technology to facilitate software development. In addition, the components are loosely coupled to enable possible extensions and Web service orchestration. For improved usability, each node is accessible via a browser. Moreover, we aim to achieve transparent activities by applying novel monitoring components for the Train requester. Finally, implemented security-related features alleviate the danger of possible data leaks.

►**Fig. 1** gives a high-level overview of the architecture components, which will be discussed in the next sections.

Trains

In general, Trains contain specific analysis tasks, which are executed at distributed data nodes—the Stations. To complete their task, a Train moves from Station to Station to consume data as an outcome of the executed analytical task. The Train requester, for example, a researcher, selects the Station sequence to be visited. The results are incrementally generated and can be anything based on the Train code. For example, the result set can contain data on an aggregated level, for example, a number showing a cohort size, which has no relationship to individual patient data of the input level, or updated parameters of a statistical model, such as a regression model that is incrementally fitted from Station to Station.

In our architecture, the analysis code is encapsulated in an Open Container Initiative-compliant (opencontainers.org) image—specifically, in a Docker image with predefined input interfaces for the data stream and output interfaces for the results, which are stored inside the container itself. This design choice provides extendibility of our architecture such that data nodes can be added easily without restricting them to certain operation system (OS) requirements to run the Train. This enables a self-sufficient execution of the analytical task, which implies that no additional installation is needed. Every necessary dependency to run the code is captured within the image. Thus, algorithm designers do not have to deal with the variety of environments in the data nodes. Consequentially, the analytic tasks can be written in any programming language. Further, the Train is enriched with metadata to increase the transparency of the analytical tasks consuming sensitive data. This metadata provides, for example, information about the data the code is accessing or information about the Train creator.[20]

The Train itself is executed in a so-called Docker-in-Docker (DinD) container. With this approach, we isolate

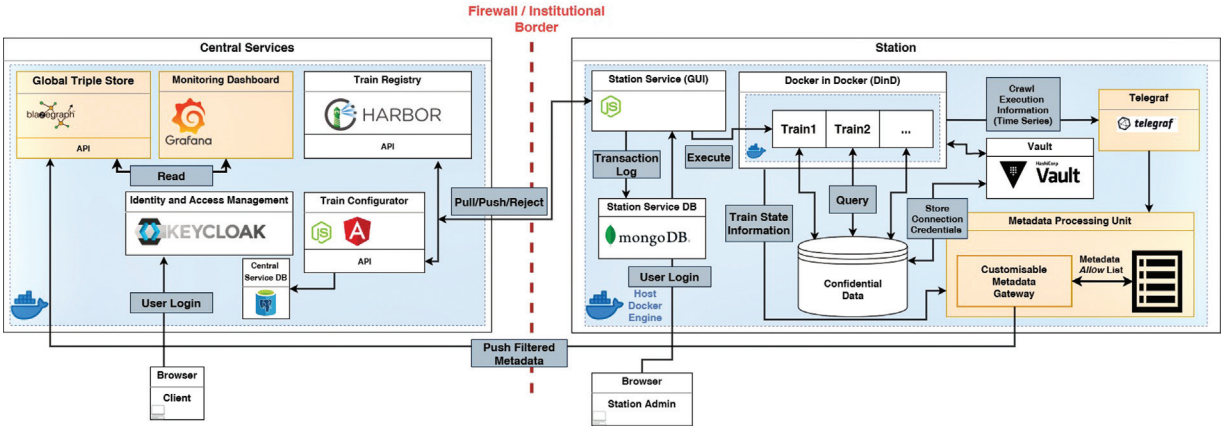

Fig. 1 Personal Health Train (PHT) architecture. This architecture consists of a central managing unit and a separated autonomous Station environment. Each subcomponent is accessible via a browser. All in all, we have four communication channels between the Central Service (CS) component and Station(s): Pull, Push, Queue Request, and Reject. Based on these commands, our Train Configurator manages the synchronisation of the Train Registry (Harbor) and the Central Service database. In parallel, this architecture includes monitoring components (orange). On the Station side, we crawl the necessary information about the local transactions. Via a gateway, the metadata/monitoring data are pushed to a global metadata store and visualised by Grafana.

the Train execution from the host Docker engine and create *sandbox-inspired* runtime environment, which involves another layer of security.

Trains can have several states in their lifecycle, which adds transparency about the state-flow of a Train itself and provides information about the Train status to the users. Possible states are illustrated in ►**Fig. 2**. First, a Train Class is created and stored in a so-called App Store (Train Class repository) after the domain community examined the Train to prohibit malicious code executions at the Stations. If a researcher wants to conduct a data study, the researcher selects a suitable Train Class and a new Train Instance is created. According to our workflow definition, a Train can be in an idle state if it waits in the queue for being pulled by a Station. After the transmission, the Train remains in the idle state until it is transferred to the running state if the Station executes it. If the Train execution at a Station was successful, the Train is pushed back to the repository and the workflow starts again at the next Station defined in the route. In unsuccessful cases, the Train is also pushed back but for debugging purposes and the sequence stops. Further, the statechart (►**Fig. 2**) includes corresponding states if the

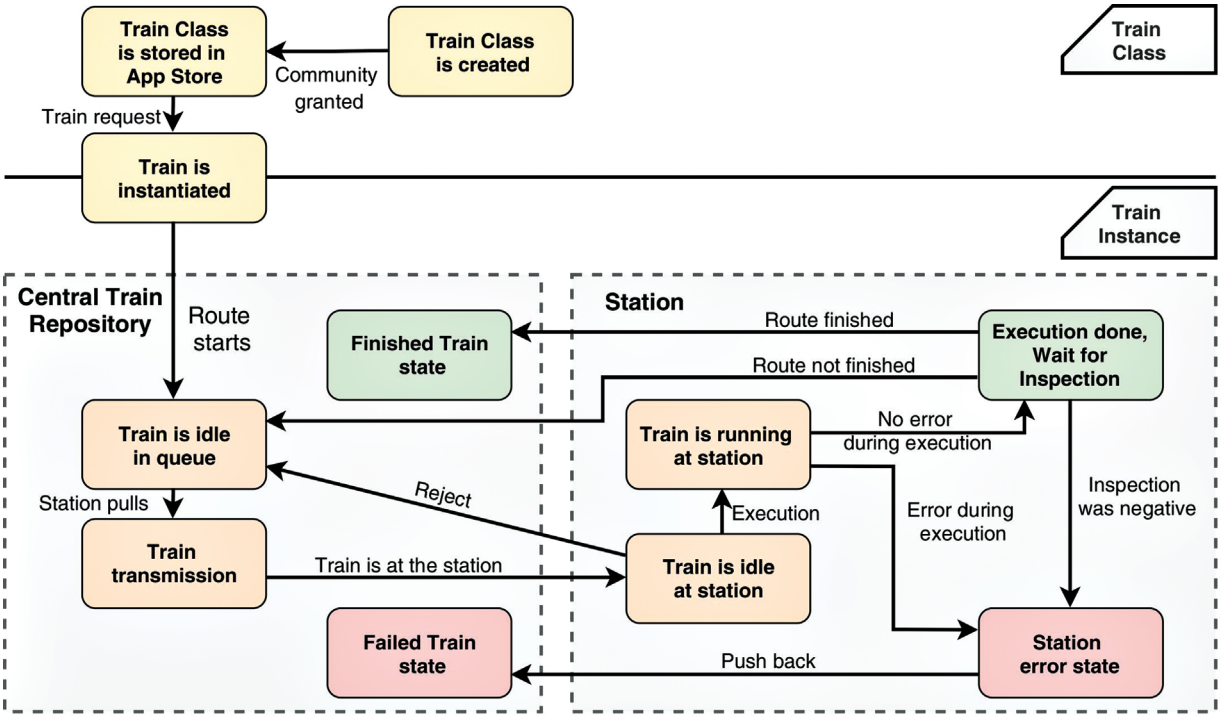

Fig. 2 Train lifecycle diagram. In general, we have two state types in our workflow. The first (yellow) states represent the states of the Train Class in the App Store. If a researcher requests a Train, the Train is instantiated and is following the states in the lower part of the figure. The lower part represents the actual Train circulation in our Station network.

Station sequence was processed successfully or an error occurred. Finally, the Train requester is able to inspect the results. The advantage is that at any point in time, the Train requester is aware of the current state, which is leverage for the usability of such a network.

Station

Stations are the nodes in the distributed architecture that hold confidential data and execute analytic tasks (►Fig. 1). Each Station is registered in a Station Registry and acts as an autonomous and independent unit. In the distributed architecture, the communication is designed to be one-way, that is, Stations are actively polling the CS if there are Train requests waiting to be executed. In contrast, the CS does not have an active channel to the Station such that the Station admins or curators of privacy-sensitive data have at any time the high-level sovereignty of any pull, execution, or push activities affecting the corresponding Station. They decide independently to accept or reject requested analytical tasks. ►Fig. 3 presents this policy. After the Train is queued and the Station obtained the list of all waiting Trains, the Station admin is able to reject a Train, for example, due to doubts about the data usage or a lack of capacity. Further, after passing the first quality assurance, the Train is executed. As the last step, the Station admin inspects the Train results. The Train can be rejected if the result set contains confidential information. Hence, it is ensured that only privacy-conform results leave the Station boundaries.

The Station has two main components: The data source and the Station software (►Fig. 1). The Station can hold the data itself or provides an access point to the sensitive data. The main task of the Stations is the execution of the containerised analytic algorithms. Therefore, every Station communicates with a local Docker engine to execute a Train. This execution consists of five steps, that is, pulling an image from Train Registry to the local machine, creating and starting the container of the corresponding pulled Train image, committing the container to create a new image from the container's changes, and pushing the changed image back to the Train repository. Furthermore, since the data providers or institutions could have different authentication procedures, we do not restrict the Station software to one single authentication technology and leave the integration of the authentication mechanism to the institution. In addition, each Station provides a graphical user interface representing a management console to coordinate the Train execution cycle (see the lower part of ►Fig. 2).

Central Service

The CS (►Fig. 1) provides several monitoring and management services, for example, to define and execute the Station sequence, to secure intermediate and final results generated by the Trains, and to provide interfaces to the repositories of each operating partner in our architecture. In ►Fig. 4, we depicted a general workflow of our CS. First, it covers a so-called Train Class repository. Researchers and scientists are able to propose and store their developed analytic algorithms containerised in a base image, which is termed Train Class, and make them available to others. If the researcher requests an analytical job and defines a sequence of Stations to be visited, the CS replicates the base image and stores it in the repository of the first positioned Station in the sequence. After the Station pulled, executed, and pushed the Train back, the CS copies the new Train image into two repositories: The User repository representing the execution history and the repository of the next Station in the sequence for the subsequent Train execution.

The CS is developed as a RESTful Web service based on Express, a Node.js framework, and running as container on Docker. This service utilises APIs and webhooks to trigger the above-mentioned procedures, for example, Train pulled, Train pushed, or Train rejected. As central Train repository, we use Harbor (Harbor: goharbor.io) to store the Train images and to provide role-based and access-controlled repositories. Further, researchers are able to create analytic jobs and define the Station sequence through a graphical user interface (GUI), which is connected to a Station Registry. This registry captures every participating Station and manages required metadata about the Stations. The GUI queries (API) only those Stations for the sequence definition, for which the

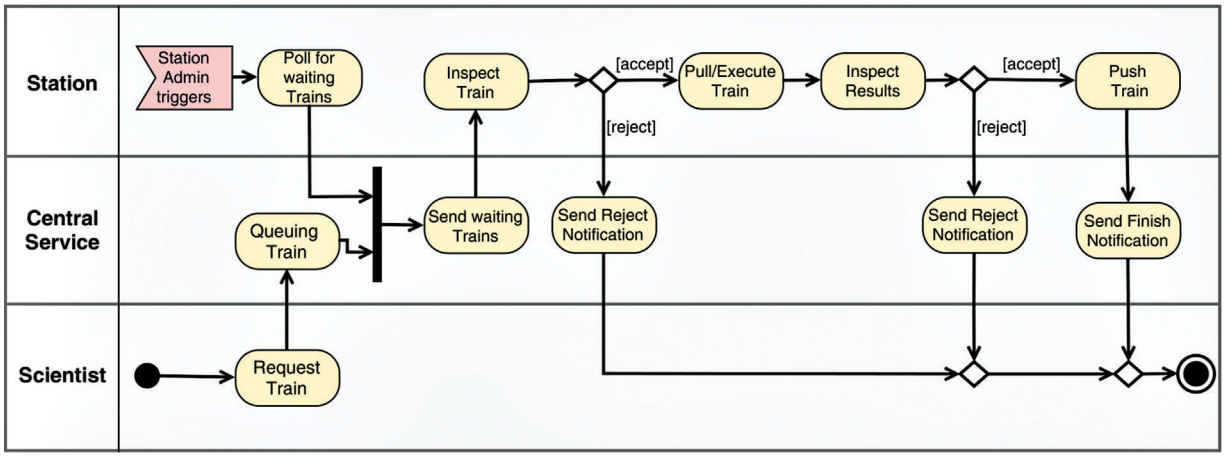

Fig. 3 Swimlane diagram for a Train request. Three instances are involved during a Train request and execution. The scientist requests the Train and the Central Service (CS) manages the whole communication with the Station. The Station has the opportunity to reject the Train before each pull/push operation to avoid malicious activities or to prohibit the return of sensitive results.

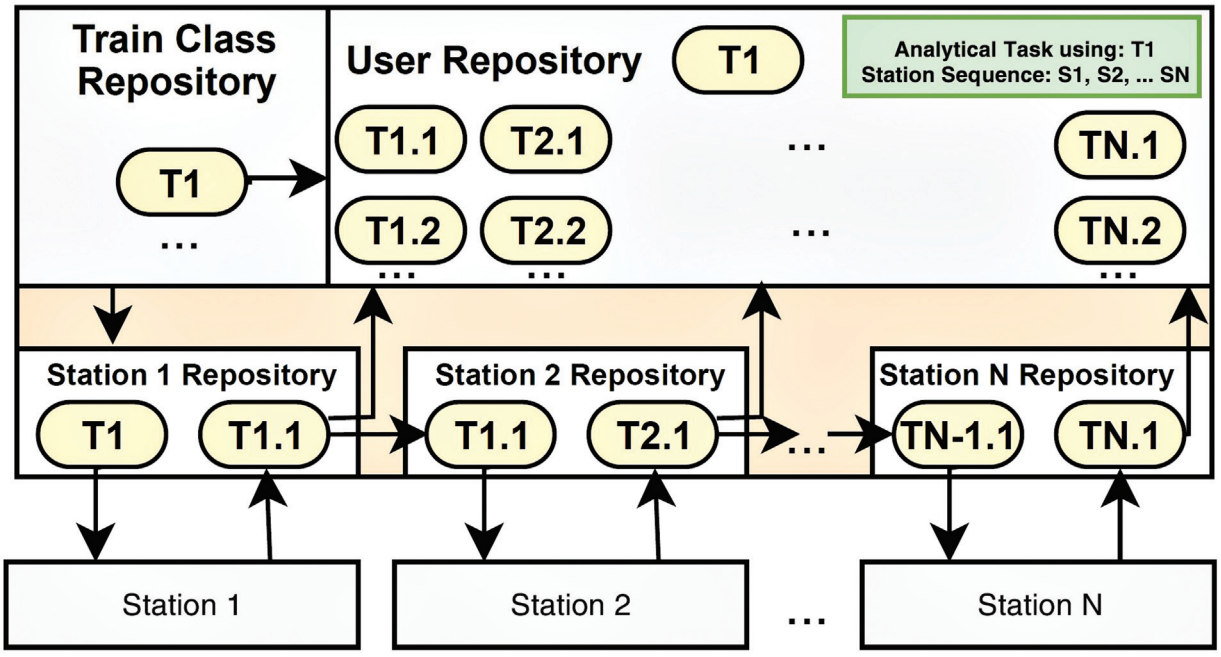

Fig. 4 Overview of the communication between the Central Service (CS) and the Stations. In general, there are three types of repositories. The Train Class repository stores the base images of each Train Class. The User repository is only accessible for the user and stores the latest results representing the history of the analytical task. Further, each Station has its own repository to get the poll information. The replication procedure is automatically done by the central unit.

researcher is authorised. This involves an additional layer of security in our architecture. The access for scientists and researchers is controlled by an identity and access management (IAM) component on the central unit. As IAM, we use the open-source software Keycloak (Keycloak: www.keycloak.org) to manage user accounts and access authorisations for components of our central services.

Monitoring Components

In addition to the basic operative functionality of our infrastructure (e.g., Train management), we experienced rising non-transparency if the number of participating parties, especially Stations, increases. This blackbox-alike behavior of these architectures does not contribute to trust between the involved actors. For example, a Train requester might not necessarily be informed about the current Train position or state. On the other hand, the Station admin should have detailed information about each Train which arrives at the Station. To tackle this problem of lacking information, we developed a novel metadata schema, which enriches each incorporated digital asset with detailed semantics, in one of our previous works.[20] This metadata schema—based on Resource Description Framework Schema (RDFS)—is used by our monitoring components to provide descriptive information to the actors (see orange components in ►Fig. 1). Each Station has a so-called Metadata Processing Unit, which saves static metadata about the Station (e.g., Station owner, available data sets) or collects dynamic Train execution information (e.g., current state or central processing unit consumption) from the DinD engine. The dynamic data instances are gathered, converted according to the schema standard, and sent to the global metadata store, which is located in the CS. The triple store acts as a backend for the Grafana (Grafana: grafana.com) frontend, which visualises the stored information for the Train requester. We have further considered the Station's autonomy by applying a customisable filter for the metadata stream. This means that the admin of the Station still maintains control over the outgoing information. For this, we implemented deny/allow lists for the metadata stream, which can easily be tailored by the admins according to the present legal circumstances.

Privacy and Security Components

Our infrastructure follows several design principles to protect sensitive data records. We assume that the station admin, who is interacting with the Station software, is authorised to inspect and release potentially sensitive data, which has been generated in the context of the Train execution (e.g., a query result or model parameters). However, the admin's authority is limited and is only valid within the institutional borders. Therefore, the admin must not see the results of the preceding stations. The admin further should also be sensitised to the intrinsic activities of the executed Train and the files inside the Train, which will be released after the Train has left the institution. To meet these requirements, our Station software incorporates a mechanism to inspect the Train contents and visualise *added*, *changed*, or *deleted* files (►Fig. 5, left). In addition, in case the Train produces query results, the admin is able to audit the file contents themselves. The software detects the changes and only visualises data, which is relevant for the current station by simultaneously hiding information from other stations (►Fig. 5, right). This

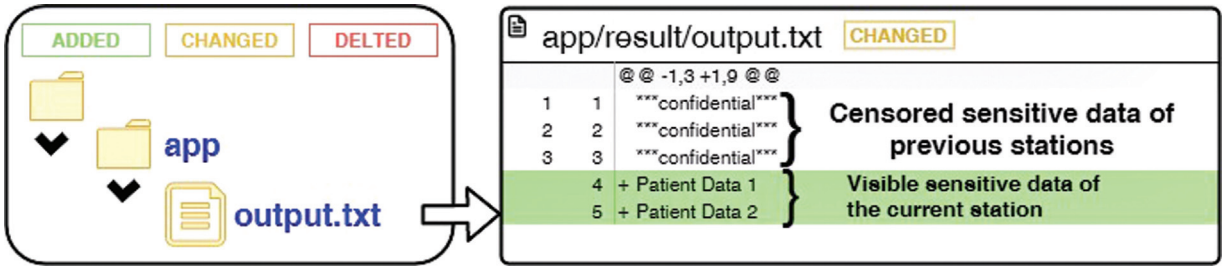

Fig. 5 Censorship of intermediate results. For query results, the Station software detects amendments of the executed Train and hides previous query outputs for the admins of the subsequent Stations. This feature guarantees that actors can only inspect data, which they are allowed to review.

feature enables increased transparency of the contents but masks accordant data as well to make first steps toward data privacy.

Additionally, our infrastructure follows a private-public key encryption policy, which has been depicted in ►Fig. 6. First, we assume that our CS is considered as (semi-)trusted. We further designed the encryption workflow in a way that no Train is stored in a decrypted format and that only the dedicated recipient can decrypt the digital assets, that is, the Trains. This approach especially involves the usage of private and public keys for each participating party: Train requester, CS, and Stations. According to our Train lifecycle (►Fig. 2), the Train requester instantiates a Train instance, which is encrypted by a symmetric key. This symmetric key is generated for each Train request ad hoc. In a second step, the symmetric key is encrypted by the public key of the first station. With this envelope encryption, the Train is securely stored in the Station's repository until the corresponding Station pulls it. After the Train transmission, the Station reversely decrypts the Train, executes it, and re-encrypts it with the public key of the CS. Then, the CS re-encrypts the Train content with the public key of the next Station. This methodology enables potential error handling, which might be inefficient if the Station encrypts the Train with the public key of the succeeding Station. Due to this reason, we assume the CS as (semi-)trusted such that we remain able to act and redirect a Train in case a Station failure occurs during the Train circulation. In the end, after finishing the Train route, the final Train including the encrypted aggregated results is stored in the requester repository such that only the requester is able to inspect the results.

After presenting our main design principles, we apply our DA infrastructure to a sample data use case in the next section.

Analytical Tasks

In this section, we apply our PHT implementation to one use case as a proof-of-concept approach. We start with the introduction of the experiment setup including the data distribution and the model selection. Finally, the next section presents the results of the performed model training. Note that we assume that our Trains will always be accepted by the Station admins. The artefacts of the model have been made available online (refer to the **Supplementary Data**). Another use case study—including lightweight statistical analysis and complex ML tasks—using our architecture can be found in the work of Mou et al.[12]

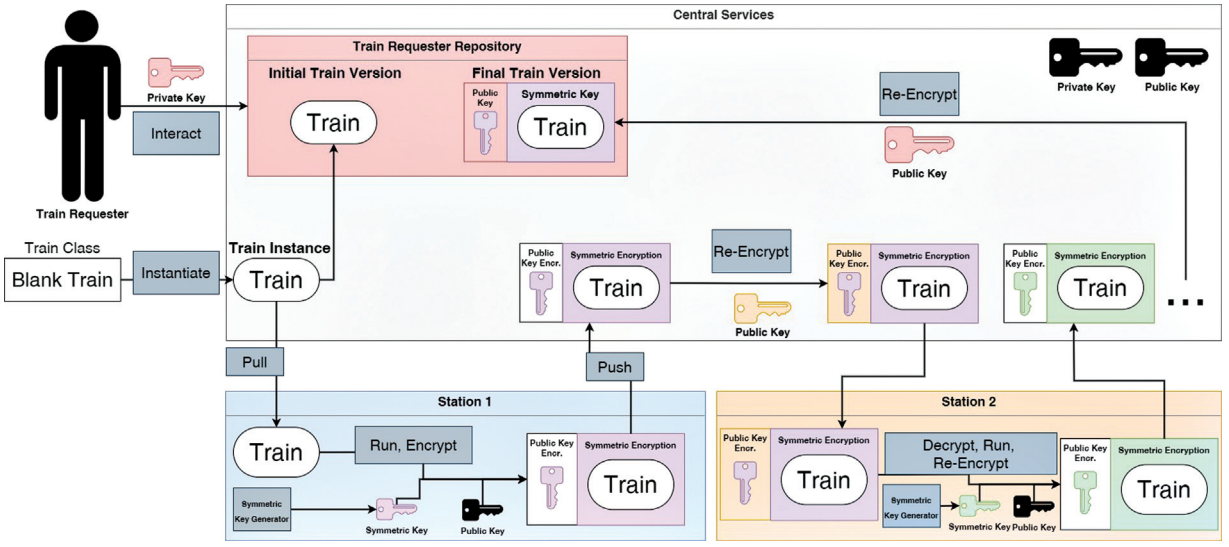

Fig. 6 Encryption/Decryption workflow. In this workflow, we assume the Central Service (CS) to be (semi-)trusted. Each entity owns a private/public key pair. For the encryption, we apply envelope encryption of a symmetric key, for example, a *session key* after each Train execution. The encryption concept itself is designed in a way such that only the dedicated recipient is able to decrypt the digital assets.

Setup

In our scenario, we perform a pneumonia detection model (Pneumo-Model) training on distributed and (class-)imbalanced data. This data set has been provided by Kermany et al.[21] It contains 5,856 X-rays including 4,273 characterised as depicting pneumonia (labelled as 1) and 1,583 normal images (labelled as 0). Thus, the data set exhibits a recognisable class-imbalance, which might affect the model training. As proposed by Kermany et al,[22] we used 5,232 instances for training (including 10% validation) and 624 instances are held out for testing after each training procedure at each Station. We split the initial data set into three equally sized chunks and distribute these chunks to our Stations S1, S2, and S3. We selected a convolutional neural network with two 3×3 (32 channels) convolutional layers, each followed by 2×2 max pooling, a 128 units fully connected layer (Rectified Linear Unit activation function), and a final sigmoid output layer. The objective function is the binary categorical cross-entropy function. Finally, we define our Train route as a cycle. The Train, encasing Pneumo-Model, visits each Station three times.

Results

Overall, we have conducted the model training twice with different initialisations and tested our models with respect to the above-mentioned test set after each training routine. We obtained two training histories and the final model statistics, which are depicted in ►Fig. 7.

While the blue model does not exceed the performance of 62.5% due to the present data skewness, we see a constant increasing accuracy of the red model, which ends up with a final score of approximately 80% correct predictions. Further, the model reaches two plateaus during the training process. The first plateau is during the third training at Station S3. This training procedure appears to be redundant since there is no further improvement in accuracy. After the fourth model transmission, we again recognise an increase in performance until the accuracy reaches the second plateau.

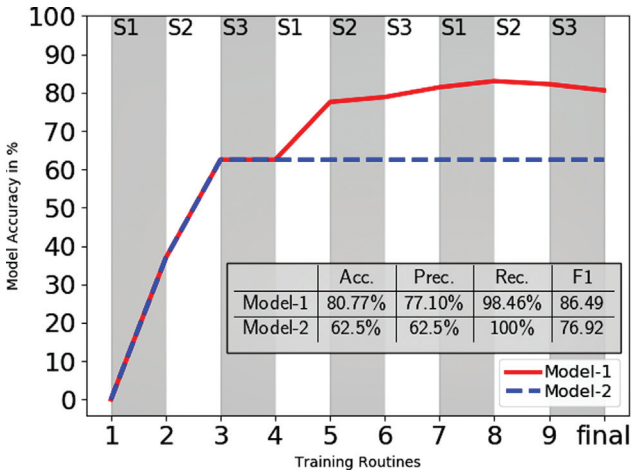

Fig. 7 Two training progresses of the pneumonia detection model Pneumo-Model with different model parameter initialisations. We perform a threefold sequence repetition of our Stations S1, S2, and S3 to train our model, which results in nine model transmissions/training routines each. Each line indicates the model performance progress of each model with respect to our test set. While the blue model training fails, we see that our architecture is able to train the red model, which reaches acceptable performance as the quality metrics indicate.

Discussion

After presenting our architecture and a proof-of-concept use case demonstrating the capabilities, we compare our implementation with the state-of-the-art with respect to functionality, usability, and privacy. We also elaborate on how our architecture complies with legal constraints and discuss additional work, which should be done to reach the full compliance with privacy regulations.

Comparison to other Technologies

In the "Background" section, we have presented several approaches that have been associated with DA during recent years. First of all, we have designed our infrastructure according to one of the two learning paradigms, which has been the sequential approach (incremental learning). Due to this asynchronous or nonparallel learning scheme, we are able to involve another layer of security, which is the Station admin's interaction (e.g., pull or push) with the software. Therefore, it is guaranteed that only authorised data accesses and analyses are allowed to be conducted by the Station software. Further, our infrastructure has the structural capabilities to additionally provide interfaces for the FL paradigm due to the flexibility of the containerisation technology. A straightforward implementation might be that replicas of a Train are distributed to each participating Station and an aggregation unit in the CS manages the incoming data streams for the global model training (or the merging of the query results) after the Station admins have granted access to the data sources once. However, although this approach has shown its promising potential, we argue that this approach lacks applicability and contradicts with our priorly mentioned assumption about the sovereignty of the data provider. Due to its parallel nature, this learning technique requires a continuous or frequent access to the data sources to efficiently train the global model since the learning protocol is round -based—after each round the Station sends the updated local model and receives the new global model for the next training round. Unless no additional precautions are made, this approach might create potential security breaches. Such a breach can be the interception of the steady data communication between server and client once the client-side connection to the data is established.

On the other hand, a reoccurring confirmation by the admin after each update round could detect potentially malicious activities but it will significantly affect the efficiency of such a parallel learning approach due to the synchronisation overhead.

A solution could be the combination of SMPC, homomorphic encryption, and FL as it has been shown by Fang and Qian.[23] They have introduced a federated algorithm, which trains a global model based on (homomorphically) encrypted data communication. While the authors have shown the feasibility of their algorithm, they also stated

that the overhead induced by additional encryption/decryption processes affects the training performance significantly, which might be aggravated by low network bandwidth. Nevertheless, it is worth mentioning that our architecture also offers the opportunity to embed an SMPC protocol like additive secret share as it has been presented in the "Background" section. Implementing such an SMPC protocol will extend our security guarantees because it will prevent potential inferences from the result set.

As we mentioned earlier, the containerisation methodology enables almost endless possibilities for algorithm design involving different complexity levels (e.g., simple queries or complex ML tasks). The same holds for the provision of data. Our ecosystem practically complies with any data storage technology and, therefore, also with emerging standards like Fast Healthcare Interoperability Resources (FHIR). This appears to be an advantage over those solutions like DataShield as the containerised analysis code can be customised according to the present data stores and we avoid the preceding data conversion into the predefined data format (e.g., Opal for DataShield). However, we are facing some shortcomings with respect to the explorative investigation of decentralised data sets since it might be an essential initial step for conducting a more complex succeeding data study. While our infrastructure inevitably necessitates the containerisation of algorithms and the associated container transmission to obtain the results, DataShield demonstrates a more efficient result turnaround time since the R commands are executed on demand without prior containerisation, which enables efficient explorative data analysis. On the other hand, as mentioned above, DataShield only allows for a restricted set of functions, which excludes hands-on ML tasks. Therefore, DataShield can be considered as a complementary solution since our architecture enables distributed ML use cases as we have shown in the "Analytical Tasks" section.

Legal Aspects and Privacy

As mentioned above, when processing sensitive data, several data privacy regulations apply. In Europe, especially the GDPR represents the legal basis for analyzing sensitive or—in GDPR terms—personal data. Especially, data such as genetic data or data concerning health are *special categories of personal data* (Article 9). Therefore, such data needs special protection and has to be processed differently. This legitimates the application of privacy-preserving DA infrastructures, which meet these specific and particular regulations of this separated group of sensitive data in the health care domain. A Data Protection Impact Assessment (DPIA) has been conducted by the developers of vantage6 (vantage6: distributedlearning.ai), which is an alternative PHT implementation. Since our infrastructure follows from a top-level perspective similar data processing and access schemes, we argue that this DPIA can be partially applied to our framework. Therefore, we can consider Articles 6 and 9 of the GDPR to be invoked. Both articles state that data preprocessing is lawful if *it is necessary for the purposes of the legitimate or public interests*. However, these articles could be also invoked if a data subject has given explicit consent for the processing of his/her data (Article 6.1 (a) or Article 9.1). In this case, additional articles (for example Article 5: Principles relating to processing of personal data or Articles 12–23: Rights of the data subject) may be relevant for the data providers, that is, Stations. Therefore, additional data subject rights might apply like the right of anonymisation or the right to restrict the data processing. Due to these reasons, we consider the data providers (e.g., the Station admin) to be responsible for the compliance with these regulations and, hence, it requires a DA infrastructure, which provides necessary features enabling the data providers to meet their responsibilities.

Our architecture addresses this important role of the data providers by its Station-centric design (e.g., one-way communication), which respects the Station's sovereignty over the data and provides privacy-enhancing tools (limited command set, result set masking, encryption) to protect the personal (sensitive) data. At any time, the Station admin has the control over the data processing, that is, data analyses. Especially, the result viewer feature (see the "Privacy and Security Components" subsection) makes the whole data analysis more transparent and explicitly visualises which data will be emitted. It supports data curators with the Station monitoring and ensures that no confidential data leaves the Station.

Lastly, the versatility and customisability of the containerisation technology facilitate the compliance with on-premise regulations and the *Principles relating to processing of personal data* (Article 5). These principles include, for example, the *data minimisation*, *purpose limitation*, or *storage limitation*, which can be easily put into practice by using container-based data provision.

Conclusion

In this article, we have shown a DA infrastructure. In our implementation, we focused on several important aspects to gain acceptance and confidence in our infrastructure.

First, we push the development of our Station software running on the data provider side into the foreground and align the development of other components according to requirements and design decisions, which originate from the Station software (e.g., data sovereignty, control over the Train executions). Thus, we intend to break down legal issues preventing a data provider to participate in our platform.

Second, we aim at a high degree of flexibility in the development of the analytical tasks and the used data format. Therefore, we use containerisation technologies to transmit the analysis code. The advantage is that the analysis code can dynamically be aligned to several data formats and the programming language is arbitrary. This means that we achieve improved practicability and usability by design.

Third, we enrich every component in our architecture with metadata and provide the metadata information via a monitoring dashboard to improve the transparency for every participant. We have shown in an example use case that our architecture is capable of managing advanced analytical tasks like data model training.

Regarding the related works done, we compared our architecture with similar solutions. We conclude that the containerisation technology offers a valuable and flexible option for conducting DA but we also emphasised that we are facing some shortcomings in efficient explorative working.

Finally, we briefly contextualised our work in the scope of the GDPR and discussed how it enhances data privacy.

Future Work

We will continually extend and improve our DA infrastructure according to runtime performance and efficiency. More, we are currently working on a secure Station on-boarding service. While our architecture preserves the privacy of query results (►Fig. 5), it remains to enable privacy-preserving ML and protect the model from so-called model inversion attacks since recent works have shown that ML models could indirectly reveal local data instances.[24–26] We partially covered this problem by using private-public key encryption but we pursue a combined solution involving more model-centric approaches.[24]

Further, we are currently developing an App-Store inspired Train repository for the community. We intend to facilitate community-driven and semi-automated Train reviews to detect malicious code, which might increase the trust in our ecosystem.

Moreover, a major aspect is record linkage, in particular, settings in which multiple Stations comprise data for the same patient. This is the case in which hospitals are very close, for example, multiple hospitals within the same region or city, but also when patients travel from hospital to hospital in the hope to get better help, for example, patients with a rare disease. Since, there is no global patient registry available mapping different, hospital-specific identifiers for the same patient, there is still the need to apply other approaches allowing to interrelate horizontally distributed data for the same real-world objects/subjects. Privacy-preserving record linkage is a method in this direction. Recent approaches look promising.[27] Therefore, we have started implementing such a linking approach, which will extend the described DA infrastructure. Finally, we will apply this DA infrastructure to further use cases, within and outside the medical domain.

Funding

This work was supported by the German Ministry for Research and Education (BMBF) as part of the SMITH consortium (SW, LN, MJ, YUY, TK, SD, and OB, grant no. 01ZZ1803K). This work was conducted jointly by RWTH Aachen University and Fraunhofer FIT as part of the PHT and Go FAIR implementation network, which aims to develop a proof-of-concept information system to address current data reusability challenges occurring in the context of so-called data integration centres that are being established as part of ongoing German Medical Informatics BMBF projects.

Conflict of Interest

None declared.

References

1 Chang K, Balachandar N, Lam C, et al. Distributed deep learning networks among institutions for medical imaging. J Am Med Inform Assoc 2018;25(08):945–954
2 Das A, Upadhyaya I, Meng X, et al. Collaborative filtering as a case-study for model parallelism on bulk synchronous systems. In: ACM Conference on Information and Knowledge Management - CIKM '17. New York, New York, USA: ACM Press; 2017:969–977
3 McMahan B, Moore E, Ramage D, et al. Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Artificial Intelligence and Statistics - AISTATS 2016. PMLR; 2017: 1273–1282
4 Sheller MJ, Reina GA, Edwards B, Martin J, Bakas S. Multi-institutional deep learning modeling without sharing patient data: a feasibility study on brain tumor segmentation. Brainlesion 2019; 11383:92–104
5 Su H, Chen H. Experiments on parallel training of deep neural network using model averaging. 2015. ArXiv: 1507.01239
6 Su Y, Lyu M, King I. Communication-Efficient Distributed Deep Metric Learning with Hybrid Synchronization. In: 27th ACM International Conference on Information and Knowledge Management - CIKM '18. New York, USA: ACM Press; 2018: 1463–1472
7 Sheller MJ, Edwards B, Reina GA, et al. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. Sci Rep 2020;10(01):12598
8 Beyan O, Choudhury A, van Soest J, et al. Distributed analytics on sensitive medical data: the Personal Health Train. Data Intelligence 2020;2(1–2):96–107
9 Sun C, Ippel L, van Soest J, et al. A privacy-preserving infrastructure for analyzing personal health data in a vertically partitioned scenario. Stud Health Technol Inform 2019;264:373–377
10 Shi Z, Zhovannik I, Traverso A, et al. Distributed radiomics as a signature validation study using the Personal Health Train infrastructure. Sci Data 2019;6(01):218
11 Deist TM, Dankers FJWM, Ojha P, et al. Distributed learning on 20 000+ lung cancer patients - The Personal Health Train. Radiother Oncol 2020;144:189–200
12 Mou Y, Welten S, Jaberansary M, et al. Distributed skin lesion analysis across decentralised data sources. Stud Health Technol Inform 2021;281:352–356
13 Wilson RC, Butters OW, Avraam D, et al. DataSHIELD – new directions and dimensions. Data Sci J 2017;16:21
14 Bonofiglio F, Schumacher M, Binder H. Recovery of original individual person data (IPD) inferences from empirical IPD summaries only: applications to distributed computing under disclosure constraints. Stat Med 2020;39(08):1183–1198
15 Pinart M, Jeran S, Boeing H, et al. Dietary macronutrient composition in relation to circulating HDL and non-HDL cholesterol: a federated individual-level analysis of cross-sectional data from adolescents and adults in 8 European studies. J Nutr 2021;151 (08):2317–2329
16 Zhao C, Zhao S, Zhao M, et al. Secure multi-party computation: theory, practice and applications. Inf Sci 2019;476:357–372
17 Doganay MC, Pedersen TB, Förg F, et al. Distributed privacy preserving k-means clustering with additive secret sharing. In: Proceedings of the 2008 International Workshop on Privacy and Anonymity in Information Society, PAIS'08, New York, USA: ACM; 2008:3–11
18 Stammler S, Kussel T, Schoppmann P, et al. Mainzelliste SecureEpiLinker (MainSEL): privacy-preserving record linkage using secure multi-party computation. Bioinformatics 2020:btaa764
19 Wüller S, Mayer D, Förg F, et al. Designing privacy-preserving interval operations based on homomorphic encryption and secret sharing techniques. J Comput Secur 2017;25:59–81
20 Welten S, Neumann L, Ucer YedielY, et al. DAMS: A Distributed Analytics Metadata Schema. Data Intelligence; 2021

21 Kermany D, Zhang K, Goldbaum M. Labeled optical coherence tomography (OCT) and chest X-ray images for classification. Mendeley data 2018;2(02):
22 Kermany DS, Goldbaum M, Cai W, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. Cell 2018;172(05):1122–1131.e9
23 Fang H, Qian Q. Privacy preserving machine learning with homomorphic encryption and federated learning. Future Internet 2021;13(04):94
24 Li W, Milletarì F, Xu D, et al. Privacy-Preserving Federated Brain Tumour Segmentation. In: Suk HI, Liu M, Yan P, Lian C, eds. Machine Learning in Medical Imaging. MLMI 2019. Lecture Notes in Computer Science, Vol 11861. Cham: Springer; 2019
25 Melis L, Song C, De Cristofaro E, et al. Exploiting unintended feature leakage in collaborative learning. In: Proceedings of 40th IEEE Symposium on Security & Privacy, San Francisco, USA; 2019: 497–512
26 Hitaj B, Ateniese G, Perez-Cruz F. Deep models under the GAN: Information leakage from collaborative deep learning. In: Proceedings of the 24th Conference on Computer and Communications Security, Dallas, USA; 2017:603–618
27 Vatsalan D, Christen P, Rahm E. Incremental clustering techniques for multi-party privacy-preserving record linkage. Data Knowl Eng 2020;128:101809