Automating Collaboration Handling and Management in Federated Machine Learning

Marius Schlegel*, Daniel Scheliga*, Kai-Uwe Sattler*, Marco Seeland*, Patrick Mäder*
*TU Ilmenau, Germany
{marius.schlegel, daniel.scheliga, kus, marco.seeland, patrick.maeder}@tu-ilmenau.de

***Abstract*—Federated learning (FL) enables collaborative and privacy-preserving training of machine learning (ML) models on federated data. However, the barriers to using FL are still high. First, collaboration procedures are use-case-specific and require manual preparation, setup, and configuration of execution environments. Second, establishing collaborations and matching collaborators is time-consuming due to heterogeneous intents as well as data properties and distributions. Third, debugging the process and keeping track of the artifacts created and used during collaboration is challenging, if not impossible. Our goal is to reduce these barriers by requiring as little technical knowledge from collaborators as possible. We contribute mechanisms for flexible collaboration composition and creation, automated collaborator matching, and provenance-based collaboration and artifact management.**

***Index Terms*—Collaborative Machine Learning, Federated Learning, Collaboration Management, Automation**

I. Introduction

The federated learning (FL) paradigm proposes collaborative and privacy-preserving training of machine learning (ML) models on federated data [1]. Multiple collaborators exchange local training gradients to collaboratively train a common global model, avoiding the need for shared and centrally aggregated data. As training data remains local with each participating client, this enables the training of ML models in application domains where data sharing is prohibited due to privacy, legal, and technical regulations, such as medical imaging [2], smart manufacturing [3], and credit risk assessment [4].

While the collaborative use of private data can lead to better model performance, the barriers to using FL are still high:

1) Concrete FL collaborations target a specific use case. Thus, the procedure of a collaboration is problem-specific and requires crafted setups and configurations of execution environments.
2) Bringing collaborators together and establishing a FL collaboration is typically a manual process. In particular, specific requirements have to be met, requiring checks for matching collaboration tasks and data distributions.
3) The collaborators typically belong to different domains or organizations. That makes it difficult, if not impossible, to debug the preparation and learning process and to trace the artifacts created and used.

This work was partially funded by the Thuringian Ministry of Economic Affairs, Science and Digital Society in the context of the project "Learning Products" (grant 5575/10-3).

In this context, we develop mechanisms that aim to lower these barriers for collaborative machine learning and model sharing. More specifically, we want collaborators in FL to require as little technical expertise as possible about the collaboration negotiation and learning processes. Ultimately, we aim for a *Federated Learning Collaboration and Sharing Platform* (FLCSP) that enables automated collaboration handling and management to address the aforementioned hurdles: (i) the creation and composition of collaboration workflows, (ii) the matching of collaborators when establishing collaborations, and (iii) the central control of interactions for seamless debugging as well as tracing of collaboration process steps.

In this paper, we contribute and present three mechanisms to foster flexible and automated collaboration negotiation, handling, and management:

1) a *collaboration composition mechanism* that enables the modular creation and composition of collaboration procedures based on conditions, learning tasks, and protocols for model specifications, training, optimization, as well as privacy and security measures,
2) a *collaborator matching mechanism* that matches collaborators and collaborations based on collaborator intents and collaboration requirements (metadata) as well as the fingerprints of collaborator' datasets, and
3) a *collaboration management mechanism* that captures and manages collaboration and artifact provenance traces to provide the foundation for querying and analyzing execution traces. That helps to understand the impact of procedure and configuration alterations on the resulting model metrics.

This paper is structured as follows: § II summarizes related work on platforms and mechanisms for collaborative ML, focusing on automation of collaboration negotiation, handling, and management. Next, we introduce the relevant components of our platform (FLCSP) and its underlying collaboration workflow (§ III). Subsequently, in § IV, § V, and § VI, we explain the three platform mechanisms. § VII concludes the paper.

II. Related Work

The widespread adoption of FL is inextricably tied to the support of FL frameworks and engines, such as TFF (TensorFlow Federated) [5], FATE [6], PySysft [7], FedML [8], Flower [9], LEAF [10], FedScale [11], and FederatedScope [12], which provide users with functionalities and programming interfaces to get started quickly and develop new FL algorithms

and applications. Building on such engines and frameworks, an increasing number of platforms and mechanisms have been developed in recent years to lower the hurdles and entry barriers for FL. Below, we briefly explain relevant platforms that are most related to our work and differentiate them from ours.

MedPerf [13] is a platform for benchmarking ML models in the medical and healthcare domain with a focus on the federated evaluation of ML models, distribution to clients and facilities, that enable each client to assess and verify the performance of models. Specific workflows distinguish between three container-based environments for different purposes, such as data preparation, model execution and inference, and performance evaluation. The flexible preparation and configuration of the containers is realized with MLCube, which is based on Docker and offers conventions for the creation of container images and provides easy-to-use interfaces for the execution of the tasks. Although there are similarities, MedPerf differs from our platform in terms of its objectives (i.e., federated evaluation vs. federated model training) and, thus, the scope of its mechanisms (i.e., collaborator matching and collaboration management).

OpenFL (Open Federated Learning) [14] is an open-source framework for FL, intended for use in cross-silo scenarios where data is split across organizations or remote data centers. Of particular interest here is the approach of defining collaborators, aggregators, connections, models, data, and other parameters that describe the collaborative training by means of a YAML file (called FL plan in OpenFL). This principal approach is similar to our collaboration composition mechanism. A collaboration procedure (called workspace in OpenFL) can be dockerized and, thus, an image can be distributed to collaborators so that they do not have to set up and configure the required environment. In detail, the differences to our approach lie in the scope and specific configuration options. Moreover, while OpenFL assumes that the collaborators have already been pre-selected and matched to the collaboration, we perform an extensive, two-step matching process.

Fedstellar [15] is a platform designed to train FL models in a decentralized, semi-decentralized, and centralized fashion across diverse federations of physical or virtualized devices. Fedstellar supports the creation of federations that comprise diverse devices, network topologies, and algorithms, and for which it provides a GUI frontend for experiment setup and monitoring. This approach is similar to ours, although our collaboration composition mechanism is based on a declarative configuration. Fedstellar also provides federation management by collecting logs and performance metrics to facilitate learning process monitoring. In comparison, our collaboration management mechanism captures the provenance of data, model, metadata, and software artifacts based on a comprehensive graph-based provenance model.

FEDn [16] is a modular and model-agnostic framework for hierarchical FL, which scales from pseudo-distributed development to real-world production networks in distributed, heterogeneous environments. FEDn provides functionality for status logging, monitoring and visualization of training progress, as well as for storing the resulting global models. Our collaboration management mechanism, on the other hand, captures collaboration and artifact provenance based on a comprehensive graph-based provenance model.

III. Platform

This section describes the context for the motivated mechanisms, the relevant components of our *Federated Learning Collaboration and Sharing Platform* (FLCSP) (§ III-A) and the underlying collaboration workflow (§ III-B).

A. Platform Architecture and Components

FLCSP is structured into three layers from a top-down perspective:

1) the *frontend layer*, which provides convenient interfaces for users (e.g., web-based user interface) and external developers (e.g., high-level REST/Python APIs),
2) the *services API layer*, which provides an internal low-level API of the services used by the frontend layer, and
3) the *services layer*, which provides modular services further subdivided into functional components and storage components (see also dependency relationships in Fig. 1).

In the following, we particularly describe the services of the *services layer*, their components, and their relationships. This part of the functional architecture is illustrated in Fig. 1.

The *session service* essentially comprises the *session & user manager*, which uses the user database (*user store*) to manage users, their metadata, and corresponding sessions. Each user is assigned one or more roles that define the available actions: A collaboration is administered and controlled by a *collaboration coordinator*. Participants of collaborations are *collaborators*. Platform-specific administration tasks are permitted to users with the *administrator* role. Users who consume models act in the *consumer* role. Although the mapping between users and roles could be implemented within this service, security-related functionalities are anchored within the *security service*.

Referring to the objectives of FLCSP, the *collaboration service* is the first of two core services. For the initiation and management of collaborations, FLCSP uses two types of metadata structures: *collaboration recipes* and *collaboration records*. A collaboration is precisely specified by a collaboration recipe that defines, e.g., the type of task, workflows for data preprocessing and federated training, evaluation methods, and conditions for collaboration participation, termination, exclusion, and sharing. Collaboration records are instances of recipes. The *collaboration compositor* provides functionality to compose collaboration recipes in a modular way, i.e., by importing existing recipes and recipe modules. In addition, the compositor is responsible for the generation and registration of *containers* used for the execution of FL procedures based on recipe specifications. The details of the collaboration composition mechanism are described in § IV. The *collaboration manager* is responsible for registration and management of collaboration records and *collaboration intents*. A collaboration record specifies the model type, data processing pipeline, privacy protocols, and optimization algorithm (distribution architecture and type of aggregation). While collaboration

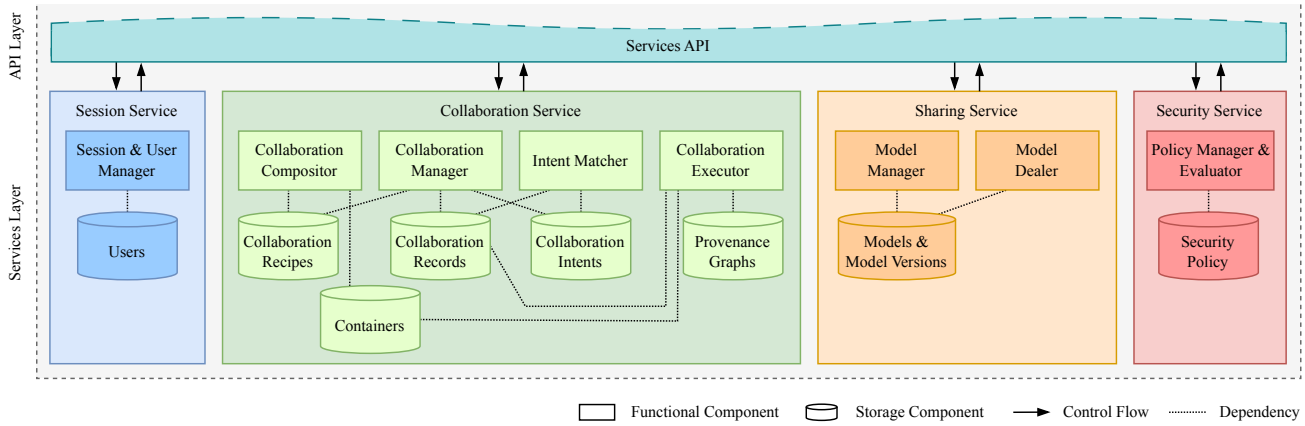

Figure 1. FLCSP's functional architecture (illustration limited to services and API layer for clarity).

records are interpreted as a kind of offer, collaboration intents are the corresponding requests. Intents are thus declarations of intent by collaborators to participate in a FL collaboration with their data and a desired task. An intent contains the metadata of a client's data so that suitable collaborations can be later found and potentially matched. The *intent matcher* matches clients' collaboration intents and collaboration records through a two-step procedure. First, collaboration records are filtered by metadata matching. After that, a more fine-grained filtering based on fingerprint matching is performed. The details of the corresponding mechanism are described in § V. The *collaboration executor* is responsible for coordinating the execution of a collaboration. This also includes the distribution of execution environments (containers) to collaboration coordinators and collaborators as well as capturing and managing the provenance of produced ML artifacts based on a graph structure [17], [18]. The details of the provenance-based collaboration management are described in § VI.

The second core service of FLCSP is the *sharing service*. The sharing service distinguishes between the components *model manager* and *model dealer*. The model manager manages and provides *registered models* and their *versions*. Each collaboration results in a registered model, where each registered model can have one or more versions. Model versions are snapshots of a registered model. Please note that depending on the number of parameters and thus the model size, it is often too costly to register a corresponding model version in each training step. Instead, a recipe precisely defines when and under which conditions a model version is registered by the model manager. For example, a collaboration may first produce a model version 1 when the training converges. New client data may then trigger a rerun of the training collaboration, the convergence of which then results in model version 2. The model manager primarily makes registered models available to internal components and collaborators who have participated in the corresponding training. The model dealer on the other hand provides market and pricing mechanisms to share models with users. It is not trivial to determine a *right* price for a model. In order to determine fair prices, the model dealer may for example perform auctions [19].

The *security service* comprises the *policy manager & evaluator* component and its *security policy*. Using hooks in each API function, the policy manager & evaluator is involved in every API call that requests service functionality. This way, (1) any interaction is inevitably controlled by the security policy and (2) decision enforcement is tamper-proof. The security policy itself is based on the paradigm of attribute-based access control (ABAC). Since ABAC enables both fine-grained and scalable specification of AC rules, it has become the de-facto paradigm for AC policy specification [20], [21], [22]. Advanced ABAC models also enable the specification of the dynamic mutability of the policy state itself (e.g., triggered by administrator workflows or changes in the system state) [23].

B. Collaboration Workflow

Aligned with the goals of FLCSP, the collaboration workflow is one of the two key workflows (see Fig. 2). It is divided into the following phases: In the first phase, the *registration phase* ❶, collaboration recipes, collaboration intents, and collaboration records are registered. Subsequently, in the *setup phase* ❷, collaboration intents are matched with suitable collaboration records. If the matching has been successful and the collaboration requirements are fulfilled, the corresponding collaboration is initialized. In the third and final phase, the *execution phase* ❸, the FL process is executed and provenance information are collected.

❶ The registration phase starts with the submission of a collaboration recipe by the collaboration coordinator to the collaboration compositor ①. According to the recipe, the compositor composes a container for later execution and registers it ②. Independently of this, a collaborator's intent for collaboration is announced by submitting a collaboration intent, including dataset metadata and fingerprint, to the collaboration manager ③. The intent is persistently recorded for later matching ④. If collaboration intents already exist, collaborations can be registered and started. For this, the collaboration coordinator requests existing intents through the collaboration manager ⑤ ⑥. A collaboration record could then be customized or tailored to the existing intents. Subsequently,

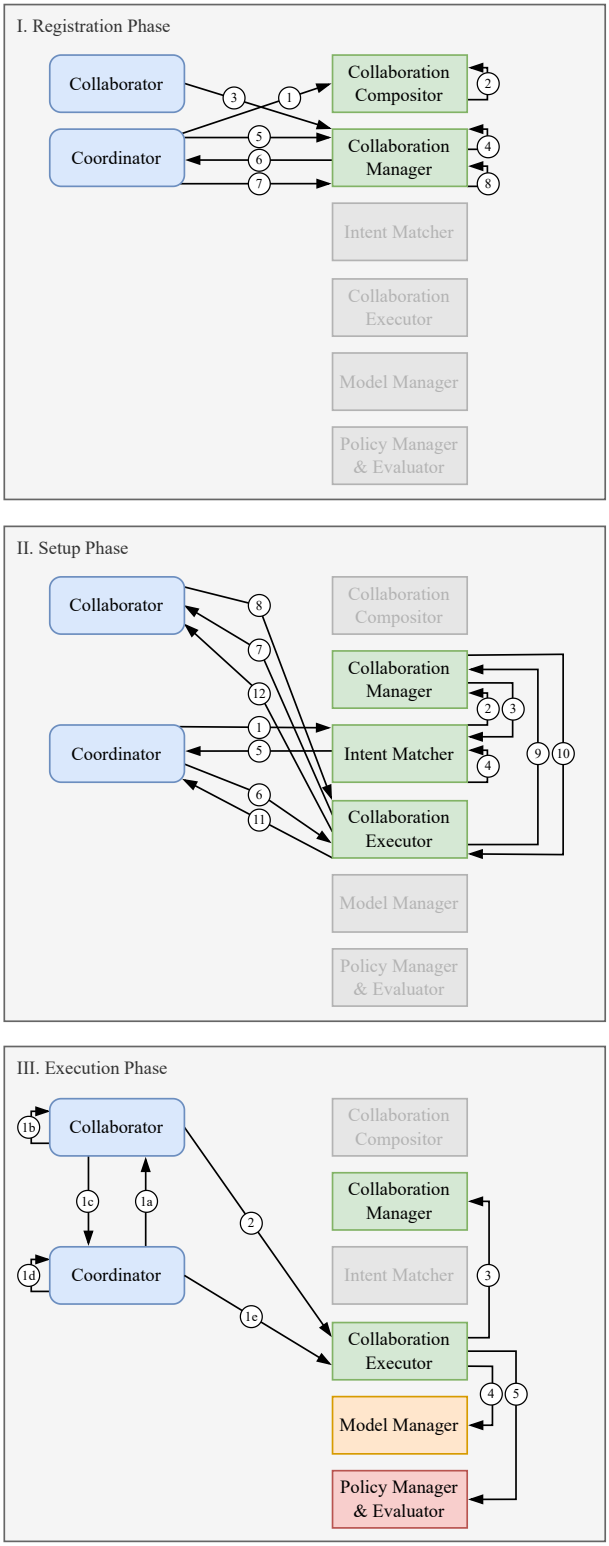

Figure 2. Workflow phase cards for the FLCSP collaboration workflow. The workflow consists of multiple phases (Roman numeral), whereas each phase consists of individual steps (Arabic numerals). In each phase card, the left side shows the users involved (i.e., users who act in a certain role), and the right side shows the functional components involved in the workflow. Components that are not involved in a phase are grayed out for clarity. Moreover, components of the frontend and services API layers and their interactions are omitted. As described in § III-A, any calls to services and service functionality are handled by the services API and controlled by the security policy.

the collaboration record is sent to the collaboration manager ⑦ and registered ⑧.

❶ The subsequent setup phase consists of two parts: the matching of intents and the initialization of a collaboration. The intent matching is requested by a collaboration coordinator based on a collaboration record ID ①. Subsequently, the intent matcher queries the corresponding collaboration record and collaboration intents based on metadata filter criteria ②. These are returned to the intent matcher ③. The intent matcher then performs the second stage of matching, i.e., fingerprint-based matching ④. After that, the intent matcher returns a list of matching collaboration intents to the collaboration coordinator ⑤.

The collaboration initialization starts with a request from the collaboration coordinator to the collaboration executor ⑥. The collaboration executor notifies the collaborators that the collaboration is about to start and asks for their participation ⑦. Collaborators then provide confirmation ⑧ to join the collaboration. The collaboration executor updates the collaboration record state and requests the corresponding execution containers from the collaboration manager ⑨ ⑩. Finally, the collaboration executor sends the execution containers to the coordinator and collaborators ⑪ ⑫.

❷ The third and final phase of the workflow is the execution phase. All platform-external communication between the collaboration coordinator and the collaborators is fully automated by the previously deployed containers. In particular, this includes the iterative execution of the federated learning process: A new training round is initiated by the collaboration coordinator ①a. Each collaborator locally trains the provided model on the own private data ①b and submits the local training state to the coordinator ①c. The coordinator then aggregates and evaluates the new global training state ①d. This may imply sending the new global training state to the collaborators for local evaluation. Each round ends with the documentation of the intermediate results by sending the global training state, including the collected provenance information, to the collaboration executor ①e. This process is iterated until a termination criteria fulfilled, e.g., until the training converges or a predefined maximum number of training rounds is reached. Once the training is completed, the final global and local training states, including the collected provenance information, are sent to the collaboration executor ②. Subsequently, the collaboration executor updates the collaboration record state ③ and registers the resulting model with the model manager ④. Finally, the model manager communicates the access rights to be granted for the registered model to the policy manager, which sets and updates the attributes of the security policy ⑤.

IV. Collaboration Composition

Each collaboration requires an execution environment that is common for all participants, consists of the same software stack, and is tailored precisely to this collaboration by an individual procedure. More specifically, the software stack of collaboration execution environments should consist of:

1) a container that is generated based on collaboration requirements and properties, and that provides a common and reproducible runtime environment for collaborators and collaboration coordinators,
2) a collaboration management and execution component, which performs the actual collaboration procedure locally for each collaborator, and
3) an artifact and provenance tracking component, which captures the artifacts produced and used and their provenance, such as data, models, and metadata, e.g., parameters and metrics.

For creating containerized environments, there are popular container engines and tools, such as Docker or the daemon-less alternative Podman. Additional tools, such as MLCube [24], [13], extend these OCI-compliant container standards with a set of conventions for creating ML-specific software container images. MLCube implements a task-based execution model: a cube typically provides a set of tasks that are executed sequentially to realize a specific workflow. For example, a cube might contain the executable tasks "data preprocessing", "model training", and "model validation". Although MLCube allows for flexible environment and workflow specification, it requires tasks to be manually invoked, which is contrary to our goals of automation and low overhead. A workaround would be to encapsulate a cube in an outer Docker container combined with a wrapper script to automate the execution of a sequence of tasks. This Docker-in-Docker approach is not recommended due to several limitations and best practice reasons. In addition, multi-container MLCubes consisting of multiple services are not trivial to implement.

Instead, we take a different approach: The starting point is a Git repository containing the implementations for the collaboration coordinator and collaborator procedures. For example, a collaborator's workflow might comprise preparation of local data and federated training. Specific procedures can be derived from existing implementations by forking the Git repository of a previous collaboration and modifying contained entry point code files (e.g., `coordinator.py` and `collaborator.py`), additional source code files, as well as project configuration and dependencies (e.g., `pyproject.toml` for Poetry-managed Python projects). To reduce the effort required to set up collaboration procedures, e.g., when experimenting with different parametrizations, collaboration recipes (`recipe.toml` as shown in Fig. 3) can be used to generate the aforementioned files. It enables the seamless integration of pre-implemented data preprocessing functions, model architectures, training protocols, or privacy mechanisms such as differential privacy [25], input encoding schemes [26], [27], and privacy modules [28], [29], [30]. Collaboration recipes are complemented by a container and image generation approach, that generates the container specifications (`Dockerfile`), which in turn serve as blueprints for building container images (e.g., using Buildah) for collaboration coordinators and collaborators.

As a popular open-source representative, MLflow [31], [32] is used for the tracking and management of ML artifacts. Its tracking component is organized around the concept of

```toml
[general]
name = "cifar10_fedavg"
description = "CIFAR10 FedAvg example"
# [...]

[data]
dataset = "CIFAR10"
path = "../data/CIFAR10"
transform_implementation = "compose"
shuffle = true
num_classes = 10

[data.train_transformations]
to_pil_image = []
random_horizontal_flip = [0.5]
color_jitter = [0.2, 0.2, 0.2]
random_rotation = [10]
random_affine = [0, [0, 0], [0.8, 1.2], 0.1]
resize = [[32, 32]]
to_tensor = []
normalize = [[0.485, 0.456, 0.406], [0.229, 0.224, 0.225]]

[data.val_transformations]
to_pil_image = []
resize = [[32, 32]]
to_tensor = []
normalize = [[0.485, 0.456, 0.406], [0.229, 0.224, 0.225]]
# [...]

[model]
architecture = "ResNet18GN"
load_weights = true
# [...]

[training]
aggregation = "fedavg"
model_trainer = "fedavg"
communication_rounds = 100
local_epochs = 2
batch_size = 64
loss = "CrossEntropy"
optimizer = "SGD"
lr = 0.001
momentum = 0.9
weight_decay = 0.0001
metrics = ["Accuracy", "BalancedAccuracy"]

[training.early_stopping]
patience = 10
delta = 0
metric = "loss"
# [...]

[privacy_options]
apply_differential_privacy = true

[privacy_options.dp_parameters]
noise_multiplier = 0.1
max_grad_norm = 10

[privacy_options.privacy_module]
kind = "PRECODE"
parameters = "default"
# [...]
```

Figure 3. Excerpt of an exemplary collaboration recipe

runs, which are executions of some piece of code. For each run used and produced data, model, metadata and software artifacts are captured based on intrusive log statements from the MLflow Python API. The information about logged artifacts and their relationships are the foundation for our provenance-based collaboration management approach. The details are explained in § VI.

V. Collaborator Matching

As described in § III-B, coordinators and collaborators register collaboration records and intents during the registration phase of the collaboration workflow. In the setup phase, potential collaborators are matched to a collaboration record

according to their collaboration intents by the intent matcher. There are four possible results of collaborator matching:

1) Both metadata and data distributions do not match, i.e., models probably do not benefit from collaborative training.
2) Metadata matches, but data distributions do not, i.e., models might not benefit from collaborative training because different data distributions can lead to unstable training.
3) Metadata does not match, but data distributions do, i.e., models might not benefit from collaborative training because of different optimization objectives and diverging feature extraction.
4) Both metadata and data distributions match, i.e., models can benefit from collaborative training.

Our proposed collaborator matching focuses on finding matchings that fall into the last category. This is achieved by a two-stage matching process consisting of *metadata matching* and *fingerprint matching*. First, potential collaborators are pre-selected based on matching metadata. Then, the collaborators' data distributions are compared based on the fingerprint matching between collaboration candidates.

a) Metadata Matching (MM): Within the collaboration intent, collaborators specify metadata of their local data that can be relevant for a collaboration. That includes descriptive metadata (e.g., intent author, descriptors, and keywords), administrative metadata (e.g., acquisition device and time of retrieval), and technical metadata, such as datatype (e.g., image, tabular, or time series), task-type (e.g., classification or regression), and annotation-type (e.g., specification of class names). Collaboration records specify the whole execution process from model selection to optimization. Besides that, they also define which metadata fields are relevant for metadata matching, such as datatype, task, target objective, and annotation format. Based on the specified relevant metadata field of a collaboration record and the registered collaboration intents, the collaboration platform pre-selects potential collaboration candidates. This pre-selection can be realized by simple attribute matching.

To illustrate our proposed collaborator matching process, we consider five commonly used deep learning benchmark datasets: MNIST [33], FashionMNIST [34], SVHN [35], CIFAR-10, and CIFAR-100 [36]. Each dataset resembles one collaborator, who formulates a collaboration intent to solve their classification problem (see Fig. 4). Furthermore, there are two registered collaboration records: *Digit Classification* and *Object Classification*. The collaboration records aim to find collaborators who intend to solve a digit and an object classification task on image data, respectively. As a result, the intent matcher would suggest the collaborators MNIST and SVHN as a pre-selection for the digit classification record and CIFAR-10 and CIFAR-100 as a pre-selection for the object classification record.

b) Fingerprint Matching (FM): Metadata matching is usually insufficient to guarantee a correct matching, because the data distributions of collaborators can still largely differ. In FL scenarios, it is well-known that model utility suffers from highly unbalanced data distributions [37], [38]. Fingerprint

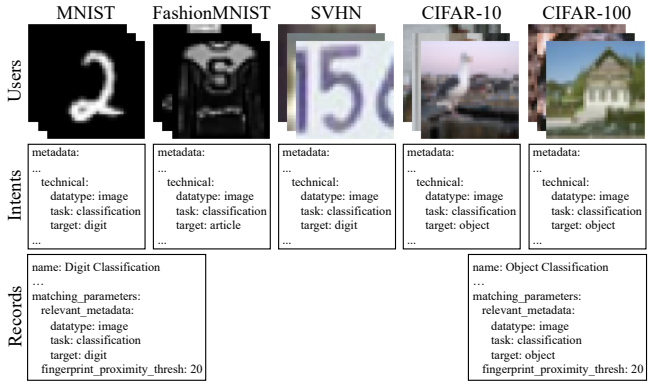

Figure 4. Example collaboration intents and records for five typical deep learning benchmark datasets.

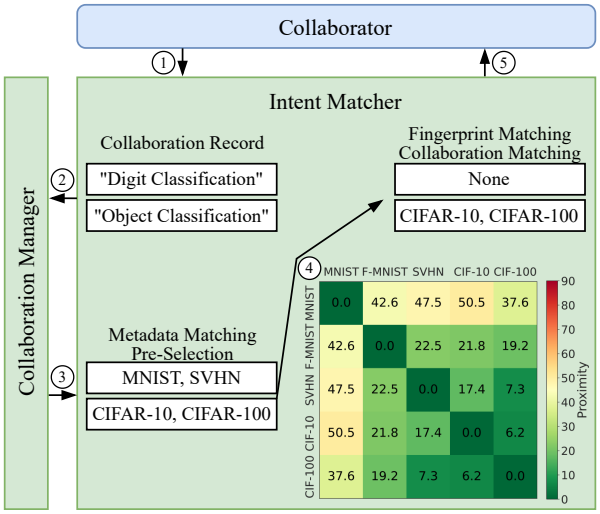

Figure 5. Example collaboration matching process, i.e., first five steps of the setup phase II (c.f., § III-B) based on the example collaboration intents and records from Fig. 4.

matching aims to mitigate this problem by matching collaborators that have similar local data distributions. To assess the similarity of different datasets without disclosing the private data, each collaborator provides a dataset fingerprint with their corresponding collaboration intent.

A dataset fingerprint is a representation of the original data distribution that satisfies four essential properties: Fingerprints should (i) capture *key characteristics* of the underlying data distribution; (ii) be *comparable*, such that similar data distributions are associated with high similarity scores, while dissimilar data distributions have low similarities; (iii) be *compact* to save communication and computation resources; and (iv) maintain the data *privacy* of collaborators. All fingerprints have to be calculated by the same protocol so that they remain comparable and can be used to estimate distribution similarities.

FLCSP uses the dataset fingerprints of all pre-selected collaborators to calculate similarities between the corresponding data distributions. The resulting similarity matrix determines which candidates are most compatible for a collaboration w.r.t. their data distribution. Based on the specifications of

Table I
Utility of trained models based on different collaborator matchings. MM/FM indicate whether metadata/fingerprint matching was used. Our proposed matching mechanism uses both MM and FM (cf. last row). We report model accuracy [%] in the format "mean ± std (absolute improvement to baseline)".

Scenario	Matching MM	Matching FM	Collaborator MNIST	Collaborator FashionMNIST	Collaborator SVHN	Collaborator CIFAR-10	Collaborator CIFAR-100
Baseline	–	–	99.06 ± 0.03	89.92 ± 0.30	89.65 ± 0.15	68.96 ± 0.24	36.18 ± 0.68
All	✗	✗	99.06 ± 0.16 (0.00)	89.93 ± 0.40 (0.01)	89.34 ± 0.12 (-0.31)	67.53 ± 0.53 (-1.43)	36.99 ± 0.25 (0.81)
Digit	✓	✗	99.05 ± 0.10 (-0.01)	–	89.84 ± 0.19 (0.19)	–	–
Object	✗	✓	–	–	89.46 ± 0.30 (-0.19)	68.70 ± 0.80 (-0.26)	38.21 ± 0.95 (2.03)
	✓	✓	–	–	–	**70.12 ± 1.17 (1.16)**	**40.06 ± 1.23 (3.88)**

the collaboration intent, a threshold determines what degree of fingerprint similarity is required to join a collaboration.

The matching method proposed by [39] could be adapted to realize a fingerprint matching phase for FLCSP. The authors apply a truncated Singular Value Decomposition (SVD) to the local dataset of each collaborator. The resulting set of principal vectors can be used as a fingerprint that "succinctly captures the main characteristics of the underlying distribution" [39]. The similarity or, in this case, proximity of two data distributions is then represented by the smallest *principal angles* [40] between the two fingerprints. The smaller a value in the resulting proximity matrix is, the more similar the distributions of the corresponding datasets.

We consider the example scenario from Fig. 5 and the metadata matching based pre-selected collaborators. In case of the digit classification record, the intent matcher would calculate the similarity of the dataset fingerprints of the MNIST and SVHN users. As a result, the proximity of $47.5 > 20$ is too large, and no matching collaborators were found. For the object classification record, the intent matcher would calculate the similarity of the dataset fingerprints of the CIFAR-10 and CIFAR-100 users. Since the proximity value of $6.2 < 20$ passes the proximity threshold check, a valid collaborator matching was found and the collaboration executor can finalize the setup phase by distributing execution containers to all coordinators and collaborators.

Comparison of Different Collaborator Matchings: We conducted a small empirical study to show the effects of different collaborator matchings on the final model utility. We train a small convolutional neural network using a simple personalized FL protocol [41]. In detail, the collaborators train the feature extraction part of the model, i.e., the convolutional layers, collaboratively and a personal classification head only locally. The models are trained for 1,000 communication rounds and one local epoch, each per collaborator per round. Local training is conducted with Adam optimizer and cross-entropy loss. After global training converged, each collaborator tunes their model locally for 5 more epochs. We repeat each experiment for 5 different random seeds and report the mean and standard deviation of the model accuracy on the test data split for each user. The results are listed in Tab. I.

To establish a baseline, each collaborator trains their model locally with only their own data and no collaboration. In the first naive matching scenario all five users collaboratively train the model ("All"). The differences in the underlying data distributions result in diverging local optima and, therefore, suboptimal model utilities. The utility of the SVHN and the CIFAR-10 users drops by 0.31 % and 1.43 %, respectively, compared to the baseline. Only the CIFAR-100 user was able to benefit from collaborative training with a utility improvement of 0.81 %.

The second scenario considers the example digit classification collaboration record from above. In this case, collaborators are only matched based on their metadata. When this collaborative training is performed between the MNIST and SVHN users, we observe only marginal improvements in model utility for the SVHN user. The unsuccessful fingerprint matching for these two users shows differences in underlying dataset distributions. Hence, these users might not necessarily benefit from collaborative training. That also indicates that it is insufficient to base collaborator matching solely on metadata.

For the object classification collaboration record, we compare the performance of fingerprint matching compared to our combined metadata and fingerprint matching approach. Metadata matching would result in the same collaborator matching as our combined matching approach. In reality, this is not always the case and might be insufficient. If only fingerprint matching is used, CIFAR-10, CIFAR-100, and SVHN would be matched as they have low proximity or similar dataset fingerprints (c.f., Fig. 5). While the CIFAR-100 user clearly benefits from this collaboration, the model utility of the other two users is decreased compared to the baseline. We attribute this behavior to the different type of classification target. While SVHN is a digit classification task, the CIFAR datasets try to classify different kinds of objects. Hence, it is also insufficient to base collaborator matching solely on dataset fingerprint similarity. If we apply our combined metadata and fingerprint matching approach, only CIFAR-10 and CIFAR-100 are matched. In this case, we can observe significant increases in accuracy for both clients, i.e., 1.16 % for CIFAR-10 and 3.88 % for CIFAR-100.

VI. Collaboration Management

As in traditional ML, collaborative ML or FL also requires the testing of different parameter combinations, model architectures, and training procedures to produce a model that performs well in terms of certain metrics. Each change and execution of a collaboration procedure results in many used and produced

artifacts, the traceability and reproducibility of which is quite challenging. These include data artifacts, model artifacts, metadata artifacts, such as hyperparameters and metrics, and software artifacts, such as data processing and model training source code, container specifications, and configuration files.

While the ML software artifacts are typically managed in repositories using version control systems (VCSs) such as Git, the management of ML artifacts is supported by so-called ML artifact management systems (ML AMSs) [32], [42], [43]. These systems allow the systematic collection and management of pipeline runs and corresponding artifacts using backends appropriate for the particular type of artifact, such as relational databases for metadata artifacts or object stores for model artifacts. A popular open source representative is MLflow [31], [32]. Its tracking component captures model parameters, metrics, outputs and other metadata. Although EMSs such as MLflow are capable of answering simple provenance questions (e.g., the best model version with respect to a given metric), they cannot be used to answer complex and composed questions that require knowledge of both EMS and VCS activities. Moreover, provenance traces cannot be queried with a declarative query language.

This motivates our provenance-based collaboration management approach. Its goal is to enable querying and analysis of collaboration provenance information, for example to understand the impact of collaboration procedure and configuration alterations on the resulting model's metrics. Based on a graph-based provenance model compliant with W3C PROV (§ VI-A), provenance-relevant information are captured by each collaborator's execution environment. These are then mapped to an instance of the provenance model, and then merged with the overall provenance graph of that collaboration (§ VI-B). The resulting provenance knowledge graph is suitable for executing predefined and user-specific queries that enable the continuous monitoring and the analysis of execution and evolution traces.

A. Provenance Model

Our provenance model maps the changes and executions of a collaboration, which are documented in Git repositories and MLflow instances to a provenance knowledge graph (i.e., a concrete instance of the model). That graph is expressive for analysis questions, such as:

- *"In which version of a collaboration procedure has metric M improved as a result of changes?"* – Particularly promising, procedure versions could be used as potential candidates.
- *"Which changes to a collaboration procedure version have improved or worsened metric M the most?"* – This question helps to find important changes and potential bottlenecks.
- *"Which changes have no positive impact on the metrics within a single collaboration procedure version or multiple versions?"* – The result points to potentially redundant changes.

The foundation for our provenance model is the W3C PROV standard [44]. It defines PROV-DM [45] – a generic provenance data model that allows domain- and application-specific representations of provenance to be expressed. PROV-DM defines the following structures relevant to this work (see Fig. 6): (i) types *entity* Entity, *activity* Activity, and *agent* Agent that are involved in producing a piece of data or artifact, and (ii) relations that relate entities, activities, and agents, such as *wasGeneratedBy*, *wasAssociatedWith*, *wasAttributedTo*, and *used*. Capturing the provenance of entities and their relationships to other entities, activities and agents results in a directed acyclic graph (DAG). All nodes and edges of this DAG have well-defined semantics yielding a specific knowledge graph. Thus, for certain uses of PROV-DM, each type (i.e., entity, activity, and agent) has at least one specialization with application-specific semantics.

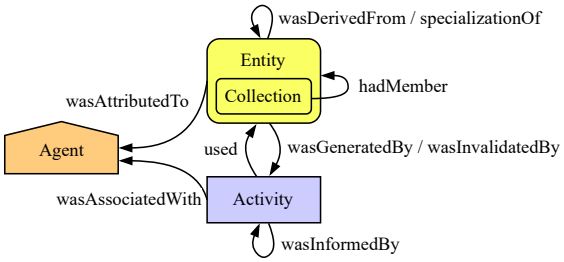

Figure 6. Overview of relevant PROV concepts.

Inspired by previous work [17], [18], our provenance model is composed of several submodels that capture the provenance-relevant events within Git repositories and MLflow instances. Types and relations of the submodels include various attributes, which are essential for later querying. Some examples are: entity attributes, such as `name` and `value` of the `Metric` Entity, and activity attributes, such as `prov:startTime` and `prov:endTime` of `Commit` Activity and `RunCreation` Activity. Fig. 7 shows a merged representation of the provenance model's submodels (attributes are omitted for clarity).

a) Git Submodels: Three submodels capture the effects that Git commits have on files' status and contents (see `Commit` Activity in Fig. 7): the creation of a new file, the change of a file, and the deletion of a file. A commit adding a file results in the following provenance information being captured:

- The file and the file revision at the point of creation (current file version) are represented as `File` Entity and `FileRevision` Entity. `FileRevision` is a specialization of `File`.
- The commit is represented as `Commit` Activity, which generated both `File` and `FileRevision`. The previous commit (`ParentCommit` Activity) is also directly linked (wasInformedBy).
- The commit author and committer are represented as `CommitAuthor` Agent and `Committer` Agent. `File` and `FileRevision` are attributed to `CommitAuthor` representing the responsibility for her content. Both agents, `CommitAuthor` and `Committer`, are responsible for that commit and, thus, are associated with `Commit`.

A commit that modifies a file has the following differences

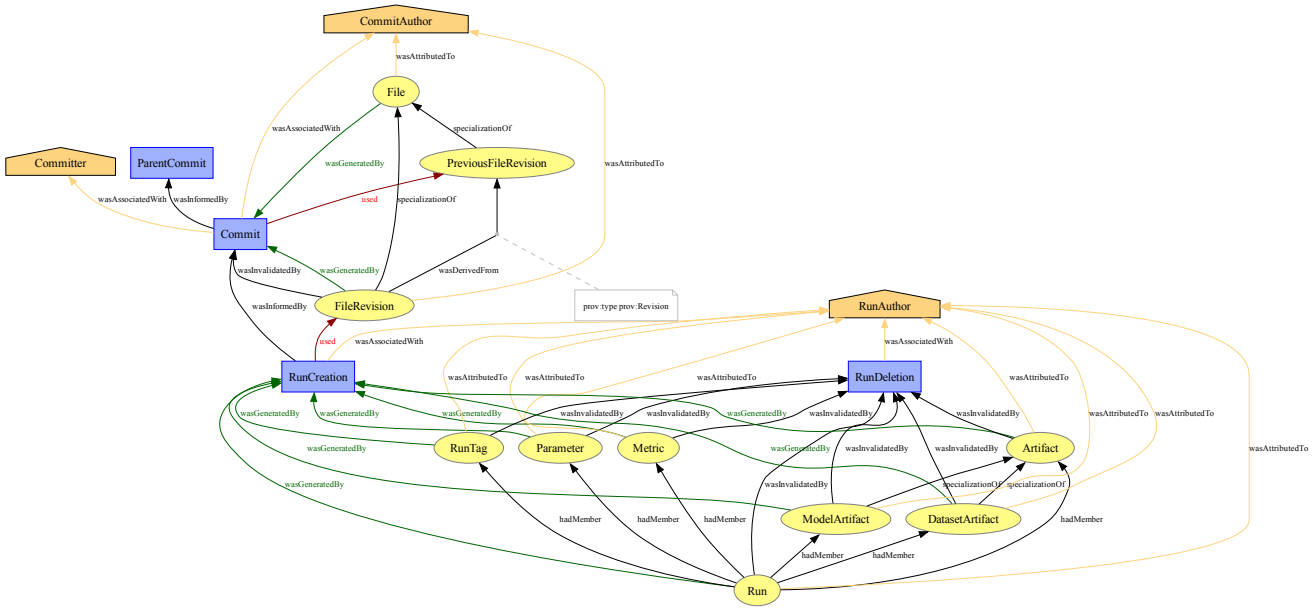

Figure 7. Merged representation of the provenance model's submodels (attributes are omitted for clarity).

regarding the captured provenance information:

- Since `Commit` used `PreviousFileRevision` (Entity) of `File`, the newly generated `FileRevision` is derived from `PreviousFileRevision`. Thus, `FileRevision` is also a specialization of `File`.

The deletion of a file by a commit results in the following provenance information:

- The current `FileRevision` is invalidated by `Commit`.

b) MLflow Submodels: Two submodels capture the effects of the creation of a new run and the deletion of a run. Compared to the file creation model, the run creation model is much more complex in terms of the provenance information collected and has some noticeable characteristics:

- `RunCreation` (Activity) uses `FileRevision` (e.g., model training code) to execute a run and, thus, was informed by the corresponding `Commit`. `RunCreation` generates `Run` (Entity), which is a collection entity, and `Metric` (Entity), `Parameter` (Entity), `RunTag` (Entity), `Artifact` (Entity), `DatasetArtifact` (Entity), and `ModelArtifact` (Entity), which are members of `Run`. The remaining relations are analogous to the file creation submodel.

Moreover, deleting runs (`RunDeletion` (Activity)) is similar to the file deletion model. The only difference is that all members of the `Run` collection entity are deleted as well.

B. Provenance Capturing & Graph Creation

The foundation for executing queries on the provenance knowledge graph is its generation based on the execution and evolution traces. Conceptually, the extraction of provenance information and the creation of the provenance graph can be done continuously, i.e., after each training round, or on demand, i.e., after the federated training process is completed. In the continuous approach, a provenance submodel graph (typically, an instance of the run creation submodel) is generated locally by each collaborator and submitted to the coordinator after each training round. In the on-demand approach, a provenance submodel graph is generated locally by each collaborator after each training round and merged with the local overall provenance graph. Upon completion of the collaboration process, i.e., federated model training, the local overall provenance is submitted by each collaborator to the coordinator and then merged with coordinator's graph as well as the other collaborator's graphs, resulting in the global overall provenance graph of that collaboration.

Integrating hooks into the MLflow's backend implementations (file store, SQLAlchemy store, REST store, etc.) makes it possible to effectively and efficiently detect relevant events and collect provenance-related information. A less intrusive way is provided by MLflow itself through a plugin mechanism that we used to modify the standard backend implementations. The evaluation of our prototype shows that the runtime overhead required for provenance graph updates, which consists of generating a provenance submodel graph and merging it with the overall provenance graph (in our tests, including at least 25 commits and 25 runs/training rounds before merging), is 18 ms at most (maximum runtime for generating and merging any provenance submodel graph). Generated provenance graphs are transferred in the PROV format and can be easily imported into a graph database. Our prototype uses the Neo4J graph DBMS, which provides the Cypher query language for querying the imported provenance graph.

VII. Conclusion

Our work aims to make FL more accessible by lowering the technical knowledge required by collaborators during

collaboration negotiation and learning processes. We argue that this can be achieved by establishing appropriate platform mechanisms for automating: (i) the setup and configuration of scenario-specific collaboration procedures, (ii) the matching of collaborators with the help of collaboration procedure requirements, and (iii) the collection and management of provenance traces. Accordingly, we developed three mechanisms for FL platforms: First, the collaboration composition mechanism that allows for the modular creation and composition of collaboration procedures, addressing the problem-specific nature of FL collaborations. Second, the collaborator matching mechanism that simplifies the formation of FL collaborations by automating the matching of collaborators and collaborations based on intents, requirements, and dataset fingerprints. Finally, the collaboration management mechanism that captures and manages provenance traces, facilitating seamless debugging and traceability of collaboration process steps. We discussed potential implementation of these mechanisms in our proposed FLCSP. Ultimately, empirical evaluation showcased the potential gains in terms of increased model utilities.

References

[1] P. Kairouz, H. B. McMahan, B. Avent *et al.*, "Advances and Open Problems in Federated Learning," *Found. Trends Mach. Learn.*, vol. 14, no. 1-2, pp. 1–210, 2021.

[2] W. Li, F. Milletarì, D. Xu *et al.*, "Privacy-Preserving Federated Brain Tumour Segmentation," in *MLMI@MICCAI '19*, ser. LNCS, vol. 11861, 2019, pp. 133–141.

[3] K. I. Wang, X. Zhou, W. Liang *et al.*, "Federated Transfer Learning Based Cross-Domain Prediction for Smart Manufacturing," *IEEE Trans. Ind. Informatics*, vol. 18, no. 6, pp. 4088–4096, 2022.

[4] C. M. Lee, J. D. Fernández, S. P. Menci *et al.*, "Federated Learning for Credit Risk Assessment," in *HICSS '23*, 2023, pp. 386–395.

[5] K. A. Bonawitz, H. Eichner, W. Grieskamp *et al.*, "Towards Federated Learning at Scale: System Design," in *MLSys '19*, 2019.

[6] Q. Yang, Y. Liu, Y. Cheng *et al.*, *Federated Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019.

[7] A. Ziller, A. Trask, A. Lopardo *et al.*, *PySyft: A Library for Easy Federated Learning*. Springer, 2021, pp. 111–139.

[8] C. He, S. Li, J. So *et al.*, "FedML: A Research Library and Benchmark for Federated Machine Learning," *CoRR*, vol. abs/2007.13518, 2020.

[9] D. J. Beutel, T. Topal, A. Mathur *et al.*, "Flower: A Friendly Federated Learning Research Framework," *CoRR*, vol. abs/2007.14390, 2020.

[10] S. Caldas, P. Wu, T. Li *et al.*, "LEAF: A Benchmark for Federated Settings," *CoRR*, vol. abs/1812.01097, 2018.

[11] F. Lai, Y. Dai, S. S. V. Singapuram *et al.*, "FedScale: Benchmarking Model and System Performance of Federated Learning at Scale," in *ICML '22*, ser. PMLR, vol. 162, 2022, pp. 11 814–11 827.

[12] Y. Xie, Z. Wang, D. Gao *et al.*, "FederatedScope: A Flexible Federated Learning Platform for Heterogeneity," *VLDB Endow.*, vol. 16, no. 5, pp. 1059–1072, 2023.

[13] A. Karargyris, R. Umeton, M. J. Sheller *et al.*, "Federated benchmarking of medical artificial intelligence with MedPerf," *Nature Machine Intelligence*, vol. 5, no. 7, pp. 799–810, 2023.

[14] P. Foley, M. J. Sheller, B. Edwards *et al.*, "OpenFL: the open federated learning library," *Physics in Medicine & Biology*, 2022. [Online]. Available: http://iopscience.iop.org/article/10.1088/1361-6560/ac97d9

[15] E. T. Martínez Beltrán, Á. L. Perales Gómez, C. Feng *et al.*, "Fedstellar: A Platform for Decentralized Federated Learning," *Expert Systems with Applications*, vol. 242, p. 122861, 2024.

[16] M. Ekmefjord, A. Ait-Mlouk, S. Alawadi *et al.*, "Scalable federated machine learning with FEDn," in *CCGrid '22*, 2022, pp. 555–564.

[17] M. Schlegel and K.-U. Sattler, "MLflow2PROV: Extracting Provenance from Machine Learning Experiments," in *DEEM@SIGMOD '23*, 2023, pp. 9:1–9:4.

[18] ——, "Extracting Provenance of Machine Learning Experiment Pipeline Artifacts," in *ADBIS '23*, ser. LNCS, vol. 13985, 2023.

[19] R. J. Kauffman, H. Lai, and C. Ho, "Incentive mechanisms, fairness and participation in online group-buying auctions," *Electron. Commer. Res. Appl.*, vol. 9, no. 3, pp. 249–262, 2010.

[20] X. Jin, R. Krishnan, and R. Sandhu, "A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC," in *DBSec '12*, ser. LNCS, 2012, vol. 7371, pp. 41–55.

[21] V. C. Hu, D. Ferraiolo, R. Kuhn *et al.*, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," National Institute of Standards and Technology, NIST Special Publication 800-162, 2014.

[22] D. Ferraiolo, R. Chandramouli, R. Kuhn *et al.*, "Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)," in *ABAC '16*, 2016, pp. 13–24.

[23] M. Schlegel and P. Amthor, "The Missing Piece of the ABAC Puzzle: A Modeling Scheme for Dynamic Analysis," in *SECRYPT '21*, 2021, pp. 234–246.

[24] MLCommons, "MLCube," 2022, https://mlcommons.org/en/mlcube/.

[25] M. Abadi, A. Chu, I. Goodfellow *et al.*, "Deep Learning with Differential Privacy," in *CCS '16*, 2016, pp. 308–318.

[26] Y. Huang, Z. Song, K. Li *et al.*, "InstaHide: Instance-hiding Schemes for Private Distributed Learning," in *ICML '20*, 2020, pp. 4507–4518.

[27] Y. Huang, Z. Song, D. Chen *et al.*, "TextHide: Tackling Data Privacy in Language Understanding Tasks," *CoRR*, vol. abs/2010.06053, 2020.

[28] D. Scheliga, P. Mäder, and M. Seeland, "PRECODE – A Generic Model Extension to Prevent Deep Gradient Leakage," in *WACV '22*, 2022, pp. 1849–1858.

[29] ——, "Dropout is NOT All You Need to Prevent Gradient Leakage," in *AAAI '23*, vol. 37, no. 8, 2023, pp. 9733–9741.

[30] ——, "Privacy Preserving Federated Learning with Convolutional Variational Bottlenecks," *arXiv preprint arXiv:2309.04515*, 2023.

[31] M. Zaharia, A. Chen, A. Davidson *et al.*, "Accelerating the Machine Learning Lifecycle with MLflow," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[32] A. Chen, A. Chow, A. Davidson *et al.*, "Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle," in *DEEM@SIGMOD '20*, 2020, pp. 5:1–5:4.

[33] Y. LeCun, L. Bottou, Y. Bengio *et al.*, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[34] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[35] Y. Netzer, T. Wang, A. Coates *et al.*, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning '11*, 2011.

[36] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009.

[37] M. Duan, D. Liu, X. Chen *et al.*, "Self-Balancing Federated Learning With Global Imbalanced Data in Mobile Systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 1, pp. 59–71, 2020.

[38] Q. Li, Y. Diao, Q. Chen *et al.*, "Federated Learning on Non-IID Data Silos: An Experimental Study," in *ICDE '22*, 2022, pp. 965–978.

[39] S. Vahidian, M. Morafah, W. Wang *et al.*, "Efficient Distribution Similarity Identification in Clustered Federated Learning via Principal Angles between Client Data Subspaces," in *AAAI '23*, 2023, pp. 10 043–10 052.

[40] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization," in *Symposium on Theory of Computing Conference (STOC '13)*, 2013, pp. 665–674.

[41] L. Collins, H. Hassani, A. Mokhtari *et al.*, "Exploiting Shared Representations for Personalized Federated Learning," in *ICML '21*, 2021, pp. 2089–2099.

[42] Weights & Biases, "Weights & Biases – The AI Developer Platform," 2024, https://wandb.ai.

[43] Allegro AI, "ClearML – MLOps for Data Scientists, ML Engineers, and DevOps," 2024, https://clear.ml.

[44] P. Groth and L. Moreau, "PROV-Overview: An Overview of the PROV Family of Documents," W3C, Tech. Rep., 2013.

[45] L. Moreau, P. Missier, K. Belhajjame *et al.*, "PROV-DM: The PROV data model," W3C, Tech. Rep., 2013, https://www.w3.org/TR/prov-dm/.