Master's thesis

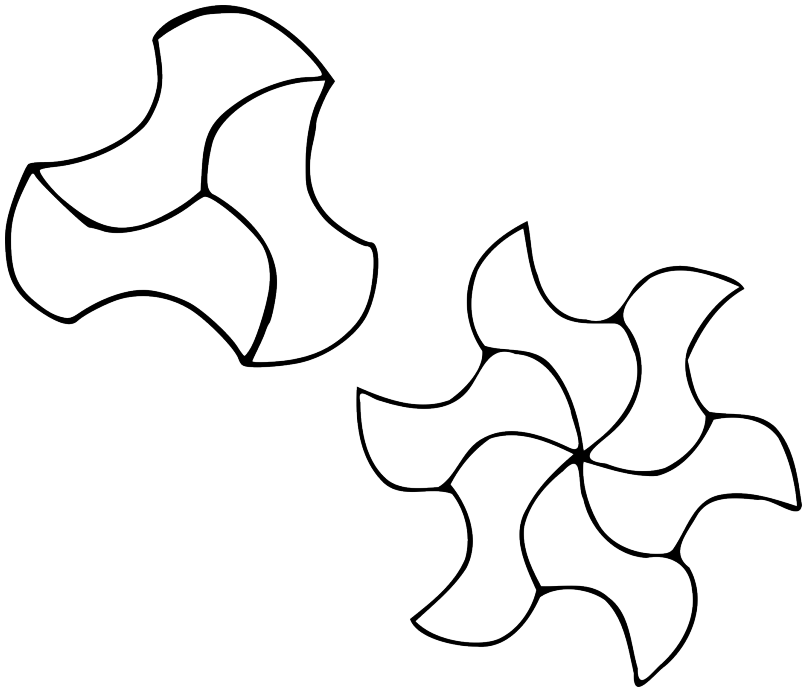
NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Ask Berstad Kolltveit

Feature Reuse in Federated Learning

Toward a Federated Feature Store

Master's thesis in Computer Science
Supervisor: Jingyue Li
June 2022



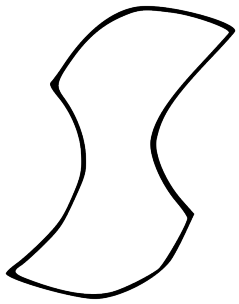
Birgitte Johanne Berstad

NTNU
Norwegian University of
Science and Technology

Ask Berstad Kolltveit

Feature Reuse in Federated Learning

Toward a Federated Feature Store



Master's thesis in Computer Science
Supervisor: Jingyue Li
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Feature Reuse in Federated Learning: Toward a Federated Feature Store

Ask Berstad Kolltveit

June 27, 2022

Abstract

Organizations deploying Machine Learning (ML) models at scale report numerous benefits of having centrally defined and reusable ML features when building new models, such as reduced Feature Engineering (FE) efforts and consistency in features in both training and operations. Federated Learning (FL) is a recent and actively researched area of distributed ML that aims to address data privacy issues in ML model training. While FL is still in a very early phase of adoption, it is reasonable to assume that it will be increasingly used as the number of Internet of Things (IoT) and other edge devices grow. Extrapolating from the reported benefits of feature reuse in centralized ML contexts, one may expect that feature reuse can achieve similar benefits for FL applications. However, current systems addressing feature reuse, Feature Stores (FSs), assume that data is centrally accessible, and thus they do not satisfy the constraints of FL. To address this limitation, the thesis poses the RQ: **How can we reuse ML features across different applications in an FL environment to increase resource efficiency on client devices?**

The RQ is answered by following a Design Science Research (DSR) methodology to design and implement a system for feature reuse in FL, the Federated Feature Store (FFS). The FFS is evaluated by first applying it to two different FL tasks, then measuring resource consumption during FE and feature ingestion for varying levels of feature reuse. The evaluation demonstrates a linear decrease in resource usage for increasing levels of feature reuse during FE and feature ingestion. The decreased resource usage implies reduced energy consumption, which is not only crucial for battery–powered mobile devices but also provides economic and environmental benefits.

The thesis results shed light on opportunities for further research into feature reuse in vertical FL, security and access-control measures for feature reuse, and design improvements to improve performance in cross–silo FL applications. Researchers are encouraged to study not only the engineering and technical benefits of ML feature reuse but also examine the environmental and economic impacts.

Sammendrag

Organisasjoner som deployerer maskinlæringsmodeller (ML-modeller) i stor skala melder om en rekke fordeler ved å ha sentralt definerte og gjenbrukbare ML features når nye modeller skal bygges, slik som ved redusert dataprosesseringsinnsats og konsekvente features både ved trening og bruk av modeller. Federated Learning (FL) er en ny teknikk innen distribuert ML som søker å håndtere utfordringer rundt privat og distribuert data ved trening av ML-modeller. Til tross for at FL fortsatt er i et veldig tidlig stadium, er det rimelig å forvente at det vil bli brukt i økende grad i takt med veksten av Internet of Things (IoT) og andre *edge*-enheter. Med utgangspunkt i de rapporterte fordelene ved gjenbruk av features i sentralisert ML, kan en forvente at feature-gjenbruk kan gi lignende fordeler for FL-anvendelser. Men, nåværende løsninger for å gjenbruke features, Feature Stores (FSs), antar at all data er sentralt aksesserbar, og de kan derfor ikke oppfylle kravene FL stiller. For å addressere denne begrensningen i eksisterende løsninger, vil denne oppgaven forsøke å besvare problemstillingen: **Hvordan kan vi gjenbruke ML-features mellom ulike anvendelser av FL for å effektivisere ressursbruken på klientenheter?**

Problemstillingen besvares ved å følge en Design Science Research (DSR)-metodologi for å designe og implementere et system for gjenbruk av features i FL, kalt Federated Feature Store (FFS). Systemet evalueres ved først å anvende det på to ulike FL-oppgaver, for deretter å måle ressursbruken under prosessering og inntak av data ved ulike gjenbruksnivåer av features. Evalueringen viser at ressursbruken i forbindelse med dataprosessering og -inntak avtar lineært med økende gjenbruksnivå. Den reduserte ressursbruken antyder et lavere energiforbruk, som ikke bare er viktig for eksempelvis batteridrevne mobile enheter, men også har økonomiske og miljømessige gevinster.

Resultatene fra oppgaven belyser muligheter for videre forskning på gjenbruk av features i Vertical FL (VFL), tiltak for sikkerhet og tilgangskontroll av features, og designforbedringer for å øke ytelsen i kryss-silo anvendelser av FL. Forskere oppfordres også til å ikke bare undersøke tekniske og ingeniørmessige fordeler ved gjenbruk av features, men også å se på miljømessige og økonomiske virkninger.

Contents

Figures

Tables

Acronyms

API Application Programming Interface. 13, 19, 29–31, 35

CFS Client Feature Store. ix, xi, 25–33, 35–38, 46, 61

CS Central Server. 5, 6, 22, 27, 31, 37, 44

DI Dataset Instance. 30–32

DSL Domain-Specific Language. 12

DSR Design Science Research. iii, v, 2, 3, 20

DSRM Design Science Research Methodology. 17, 18

ESG Environmental, Social, and Governance. 8, 45, 49

FE Feature Engineering. iii, 1, 2, 7, 15, 23, 38, 40, 42, 43, 46, 49

FEDS Framework for Evaluation in Design Science. ix, 19–22

FFS Federated Feature Store. iii, v, ix, xi, 25, 26, 28, 29, 32, 34, 35, 43–47

FG Feature Group. ix, 12–14, 25–27, 29–33, 35, 36, 40, 44, 46

FL Federated Learning. iii, v, ix, xi, xiii, xiv, 1–3, 5, 6, 15–22, 25–28, 30–32, 35–37, 39, 42–47, 49

FLC Federated Learning Client. 5, 6, 27, 28, 30–32, 37, 43

FR Feature Registry. ix, xi, 25–27, 29–32, 35, 44, 46, 49, 61, 62

FS Feature Store. iii, v, 1–3, 5, 8, 11–14, 17–21, 25, 38, 43–46, 49

FTL Federated Transfer Learning. 6

HFL Horizontal FL. 6, 16, 25, 44–46

IoT Internet of Things. iii, v, 1, 6

IT Information Technology. 8, 9, 17, 49

ML Machine Learning. iii, v, 1–3, 5–8, 11–15, 17, 18, 32, 35, 44–47, 49

RPC Remote Procedure Call. 32, 46

SE Software Engineering. 1, 3, 5, 7

SGD Stochastic Gradient Descent. 39

VFL Vertical FL. v, 6, 15, 44–46

Chapter 1

Introduction

1.1 Background and Motivation

Federated Learning (FL) is a recent technique for achieving privacy-preserving Machine Learning (ML) on distributed data [1]. ML on centralized data has already become a widespread commercial technology in the past decade [2]. Enabling collaborative model training on distributed private data, such as that generated by Internet of Things (IoT) devices, could unlock many opportunities for value creation [3–5].

However, deploying ML applications at scale has proved to be challenging from a Software Engineering (SE) standpoint [6, 7]. Much of the complexity arises because of the vast software infrastructure required for building and maintaining ML systems, particularly for handling data [8].

Organizations deploying Machine Learning (ML) models at scale, such as Uber [9], Facebook [10], and Microsoft [11], report numerous benefits of having centrally defined and reusable ML features when building new models. Some benefits include reduced Feature Engineering (FE) efforts and consistency in features in both training and operations.

Currently, the lack of solutions enabling feature reuse for FL is not a problem. FL is still an emerging technology and has rarely been applied in large-scale real-world use cases, such as in the Google Gboard [12]. However, it should be expected that FL will face similar (or more significant) data challenges as centralized ML when deploying many different applications simultaneously.

While the large-scale use cases of FL have only been realized to a minimal degree, there is reason to expect that such use cases will become more prevalent. Judging by the large amount of work on FL, the growth of IoT adoption and edge computing, and the increasing usage of ML, more widespread large-scale deployments should be expected as FL matures.

However, no work has been done on feature reuse for FL. For the centralized ML case, feature reuse is addressed with Feature Stores (FSs), such as Hopsworks [13] or Feast [14]. Existing FSs cannot be used with FL for multiple reasons:

- Current FSs are heavily focused on cloud deployment.
- Current FSs are built for centrally accessible data, contrary to the fundamental principles of FL.
- An FS for FL requires that feature data is stored locally on the clients while adhering to the centralized feature definitions.

To the author's knowledge, there has been no research to date into reusing ML features in an FL environment. As a result, the current solution is potentially redundantly computing and storing features across different FL applications. Each project would have to deploy and manage a storage mechanism for its data, resulting in resource inefficiencies from redundant computation (the same feature is computed multiple times) and storage (the same feature is stored in multiple locations). Further, it could be challenging to reuse old features when creating new models, as feature computation code may not be compatible and may be entangled with other pipeline parts or poorly documented. Lastly, it would also be challenging to discover reusable features created by other teams in the organization.

1.2 Research Question and Methodology

Given the current lack of solutions for reusing ML features in FL, the RQ of this thesis is: **How can we reuse ML features across different applications in an FL environment to increase resource efficiency on client devices?** To answer the RQ, a Design Science Research (DSR) approach is used to design and implement a proof of concept for a system that enables feature reuse in an FL environment. The system is evaluated with three experiments.

The first two experiments evaluate the system's compatibility with FL. Each experiment trains two identical models in parallel with feature reuse levels ranging from 0%–60%. The first experiment is a multivariate regression problem on the *Superconductivity* [15, 16] dataset, while the second experiment is a binary classification problem on the *Adult* [15] dataset.

The third experiment evaluates the impact of feature reuse on client resource usage during FE and data ingestion. At feature reuse levels ranging from 0%–60%, client CPU usage is measured during processing and ingestion of the Adult, Superconductivity, and *Wisconsin Breast Cancer* [15, 17] datasets, after which disk usage is measured.

The evaluation shows that the system design is compatible with FL and can be used with many different levels of feature reuse. The evaluation also demonstrated a marked decrease in client resource consumption from data processing and ingestion with increasing feature reuse levels.

1.3 Contributions

The main contribution of the thesis is a novel architecture and design of a system for feature reuse in a general FL environment, achieved by extending the FS concept. Compared to related work on feature reuse, this thesis addresses the challenge of reusing centrally defined features on distributed clients with private data. The evaluation demonstrates that increasing reuse of features decreases CPU and disk usage, which has numerous technical, economic, and environmental benefits.

1.4 Thesis Structure

The rest of the thesis is structured as follows. Chapter 2 provides background information on FL and challenges in SE for ML. Chapter 3 reviews previous work on reusing ML features. Chapter 4 outlines the DSR methodology followed. Chapter 5 presents the resulting system design and experimental results. Chapter 6 discusses how the presented design compares to the related work and its impact on industry and academia and addresses the limitations of the thesis. Finally, Chapter 7 summarizes the thesis highlights and proposes some possible directions for further work on the design.

Chapter 2

Background

This chapter will provide some background on SE for ML, FL, and FSs.

2.1 Federated Learning

There are growing concerns about the privacy of data and how well the individual's sensitive information is preserved throughout the model training process, so companies have to consider privacy-aware approaches [7], as privacy is a significant challenge in ML systems [18].

Mobile devices collect a lot of personal information about its user that could be used to train ML models to provide personalization, for example, with applications such as word prediction when typing. However, training an ML model on each device individually currently appears infeasible due to inherent constraints of mobile devices, such as limited computational resources and battery capacity. Training a fully-fledged word-prediction model would likely degrade the user experience more than personalization would improve it. However, training models in the cloud with personal data from mobile devices also appears problematic. From a privacy perspective, uploading personal data to a central server for training puts the user at risk. From a technical perspective, uploading training data could be challenging because of network constraints or because the sheer volume of incoming data to the cloud would be too much for the receiving server.

First introduced by McMahan *et al.* [1], FL is a proposed solution to the problems presented above. The technique, as originally presented, assumes a setting with many devices and works as follows. A Central Server (CS) coordinates the model training and starts by broadcasting some initial global model to all participants, the Federated Learning Clients (FLCs). The CS selects some fixed-size random subset of the FLCs for training, which then proceed to train their local model copies on a small number of training examples. The FLCs then send their model updates to the CS, where they are aggregated and applied to the global model. The updated global model is distributed to the FLCs, concluding a single round of training. Many rounds of training may be performed before a model is ready.

Table 2.1: Some characteristics of different types of Federated Learning (FL)

FL type	Characteristics
HFL	Clients have different samples with identical features
VFL	Clients have the same samples with different features
FTL	Clients may have both different samples and different features
Cross-device	Many devices which may be unreliable, may have only low number of samples per clients
Cross-silo	Few devices, more capable client hardware

The above description is a high-level overview of the FL process as presented by McMahan *et al.* [1]. Kairouz *et al.* [19] describes FL in more general terms as a collaborative ML setting setting where a CS coordinates multiple clients in achieving a learning objective while keeping their data local and private.

FL can be divided into categories based on multiple characteristics. One way to categorize FL is based on how data is partitioned among FLCs [20]. In *Horizontal FL (HFL)* or *sample-based* FL, the data is said to be horizontally partitioned, meaning that local data across FLCs share the same set of features but represent different data samples. When the data partitioning is vertical or feature-based, where data across FLCs have different features, it is called *VFL*. In vertical data partitioning, a single data sample with all its features is typically distributed between FLCs, with some common sample ID to align samples during a training round. The participants must perform what is known as *encrypted entity alignment* or *private set intersection* to coordinate which samples should be trained on for a given training round [21]. The third category of FL on this axis is *Federated Transfer Learning (FTL)*. In this case, the data can be interpreted as horizontally and vertically partitioned, meaning that a client may have features or samples others do not have.

A second categorization is based on the characteristics of the FLCs. Kairouz *et al.* [19] makes the distinction between *cross-device* and *cross-silo* FL. The FL process, as described by McMahan *et al.* [1], illustrates a case of cross-device FL, where the FLCs are highly numerous and may be unreliable. In this case, communications cost is a big challenge, which may even become a bottleneck during training due to high-dimensionality model updates, communications bandwidth limitations and unreliable networks [22]. On the other hand, cross-silo FL is the case of having a relatively small number of reliable FLCs, typically representing collaborating organizations or geo-distributed datacenters.

While there is no lack of proposed application domains for FL, such as mobile edge computing [22], IoT [4], smart retail [20], urban computing [23], [20, 22], there are fewer reports of real-world applications. Some highlights include the next-word prediction model in the Gboard [12], outcome prediction for COVID-19 patients [3], browser history suggestion ranking [24], and content recommendation [25].

2.2 Data and Software Engineering Challenges for Machine Learning

Developing ML systems has become relatively easy and cheap while maintaining them over time is difficult and expensive [8]. ML systems are inherently data-dependent, causing many SE challenges in addition to the existing code challenges found in traditional software systems. Data-related challenges are some of the most frequently cited challenges of ML systems in the literature [6, 26, 27] and arise because data is inherently more challenging and complex to deal with than code, in terms of discovery, collection, access, management, and versioning [8, 11].

Typically, raw input data cannot be fed directly to ML models but has to be cleaned, processed, labeled, and transformed, which is laborious and time-consuming [18]. *Data integration*, the process of joining data from multiple sources to create datasets, increases the difficulty even further, as different sources may use different semantics, conventions, and schemas [6, 7], which may even change over time without the data consumers' knowledge [8, 28].

Data consistency across model training and inference is another potentially challenging issue. Suppose data transformations are implemented separately for use in training and inference. In that case, any differences may lead to incorrect behavior [29], but applying the same data transformations for training and inference data can impose significant maintenance effort when done manually [18]. Ideally, the same data processing implementation should be used during training and serving [30].

Data needs to be versioned to facilitate model provenance and reproducibility and is also essential for system maintenance [6]. Managing existing data has proven to be a difficult part of ML system maintenance [26]. Configuring and versioning data is essential for maintenance, and includes versioning of both data and metadata [6].

Discovering data can be a significant challenge in larger organizations [7], which may have several negative consequences. Knowing what data is available is essential for understanding what potential ML solutions can achieve. Undiscovered feature data could lead to redundant cleaning, Feature Engineering (FE), and computation, which increases the overall time spent on a modeling problem [31].

Data dependencies cost more than code dependencies for multiple reasons. The data dependencies may be unstable, meaning that the data's shape, statistical properties, or semantics may change over time without the consumers' knowledge [8, 28]. Data scientists often exert much manual effort over time to maintain data sources for multiple deployed models [32]. Data pipelines that are created ad-hoc, as is typical in exploratory phases of a project, increase in complexity over time, making feature reuse across pipelines complex [29]. For these reasons, data pipelines are quite costly to build and manage [30]. It stands to reason, then, that being able to reuse already engineered features by accessing them from a centralized location would be a significant advantage at a certain scale.

2.3 Feature Reuse and Feature Stores

An organization deploying many models may experience significant overlap in the features consumed by different models [9]. However, data scientists and ML engineers often recreate the same features for different models, resulting in duplicate engineering efforts, superfluous feature computations, and a higher total cost of maintenance [31]. Sharing and reusing features would reduce the redundant effort in creating the same feature multiple times across different teams and could provide significant value in organizations with many models and features, such as Uber [30] and Microsoft [11]. Patel [31] names this the *democratization of ML features*, i.e., making features available, discoverable and reusable across an organization.

The FS concept is a proposed solution to many of the challenges mentioned in Section 2.2, such as discovery, access, management, and versioning, and—pertinent to this thesis—it also addresses feature reuse. The idea was first introduced by Uber [9, 28], and later adopted by other large organizations like Facebook [10], Doordash [33], Airbnb[1], Spotify[2], and LinkedIn[3], but has not been studied to any great extent in the academic literature. An FS is a software component that centrally stores, manages, versions, and serves features for use during both ML model training and inference in an organization [28, 34]. It can significantly reduce both the model building time and the time and processing required to retrieve and compute features [34].

2.4 Green Information Technology

Computers and other Information Technology (IT) infrastructure have a significant environmental impact throughout their lifecycles [35]. From production, which consumes both electricity and raw materials and generates hazardous waste; to use, which consumes much electricity; and finally to disposal, which often means polluting and contaminating the environment with electronic equipment. Green IT is the study and practice of environmentally sound IT throughout the whole lifecycle of IT equipment, minimizing negative environmental impact. One area of green IT is energy–efficient computing, which can be achieved to a considerable extent through improved software design [36]. In the world of ML, feature reuse appears to be an opportunity for eliminating computational waste through software design.

In addition to ethical reasons, other incentives for businesses to move toward green IT can also be found. Environmental concern is an essential part of the financial Environmental, Social, and Governance (ESG) criteria, a set of investment criteria for sustainable investment practices. Studies suggest that a company's focus on ESG factors positively affects its corporate financial performance

[1] https://databricks.com/session/zipline-airbnbs-machine-learning-data-management-platform
[2] https://engineering.atspotify.com/2019/12/the-winding-road-to-better-machine-learning-infrastructure-thro
[3] https://linkedin.github.io/feathr/

[37]. Eliminating the waste of hardware and energy also reduces capital and operating expenditure. Further, reputation, customer demands and expectations, and regulation and legislation have been identified as key drivers for corporate sustainability [38]. Thus, businesses' adoption of green IT can be motivated not only from an environmental perspective but also from legal, economic, and publicity perspectives.

Chapter 3

Related Work

This chapter will discuss the related work on reusing ML features. While many solutions for feature reuse exist[1], they are all based on centralized data and are limited in their differentiation. Hence, only a few examples will be discussed in detail.

3.1 Michelangelo Feature Store

Li *et al.* [9] provide the first published description of an FS as part of the *Michelangelo* system, the internal ML platform at Uber. It was found that enabling feature sharing across teams and projects was highly valuable, as many modeling problems required similar features [30]. The system is designed to be used at a large scale, with hundreds of models being constantly retrained and redeployed. The platform combines feature computation and feature storage to ensure that all features are computed by the same code path.

The authors of [9] identify some distinct challenges for batch usage (for model training and batch predictions) and real-time usage (for real-time predictions), leading to their choice of a dual-database architecture, as seen in Figure 3.1. One of the issues is the difference in performance characteristics required for training/batch prediction versus real-time prediction. Model training and batch prediction require efficient bulk access, while real-time prediction serving requires low-latency access to individual records. Another issue is that query patterns are different for training or batch prediction and real-time prediction. The variability in usage characteristics leads to having a database with performant bulk access, called the offline store, storing historical features, and a database with low-latency point access called the online store, which stores real-time features. Historical features from the offline store are imported to the online store after each feature computation job, letting models serving real-time requests access historical features with low latency. Features are computed differently depending on their requirements for recency: Near real-time features are at most 5 minutes old and

[1] `https://www.featurestore.org/` lists 27 existing FSs as of writing this.

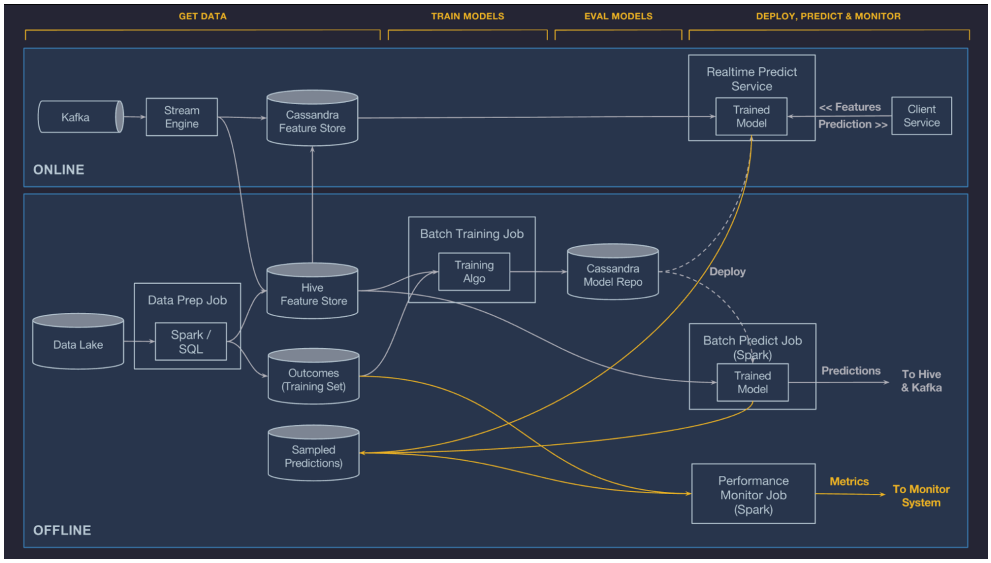


Figure 3.1: *Michelangelo* platform architecture [30].

are computed from streaming data sources, while historical features are based on data from a longer time span and are computed in batch jobs.

Feature Groups (FGs) are the atomic unit for storage, and consist of related features that are convenient to compute and access together, optimizing storage and access patterns. Each FG is a table with individual features as columns. An FG is defined by a YAML file with metadata and source files for use during computation, all of which are version-controlled. The metadata from the YAML file is published in a feature directory, which is a browsable registry of all available FGs. Deployed features should not have their semantics changed, and feature deprecation requires notifying all existing users, which are known, as feature usage is logged.

Consuming features from the FS is done by simply referencing the canonical feature name in the model configuration, and the *Michelangelo* platform takes care of joining the correct data for training, batch prediction and online predictions [30]. As features from the FS may not be in the exact format required for a particular model, a Domain-Specific Language (DSL) was developed to select, transform and combine features as they are sent to the model. The DSL expressions form part of the model configuration, such that it is versioned together with the model and the same transformations are applied during both training and prediction.

3.2 Hopsworks

Hopsworks [13] is an open-source, horizontally scalable platform for building end-to-end ML pipelines [29] [39]. It features a collaborative environment for

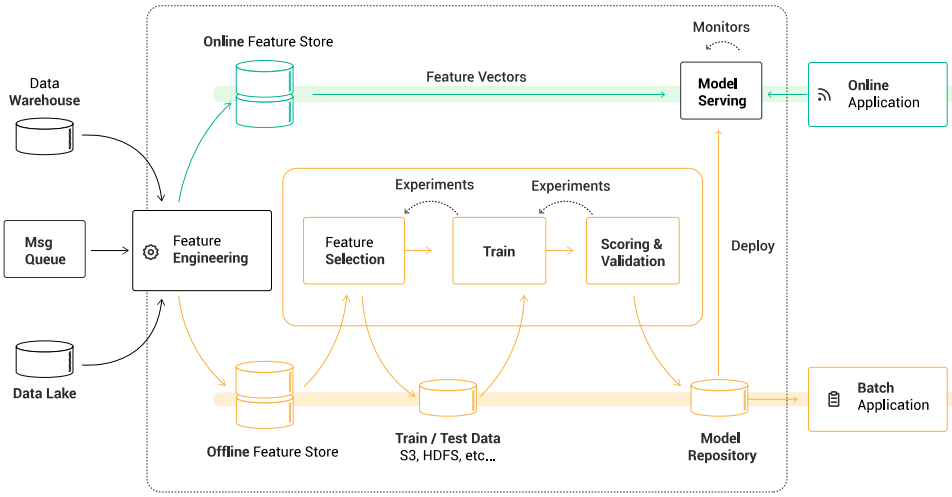


Figure 3.2: Hopsworks architecture. Taken from `https://docs.hopsworks.ai/`.

developing and deploying ML models to production and supports role-based access controls. Models can be served, managed, and monitored in production with Hopsworks. A high-level architecture diagram can be found in Figure 3.2.

An important component of Hopsworks is the FS, which computes, stores, and serves engineered features for use during both training and inference and thus acts as a central management component for all feature data in an organization. It supports versioning, documentation, access control, and time travel for features, which allows looking up the value of a feature at a specific time [40] [41]. Similar to Li *et al.* [9], the FS is based on an offline store for training and batch use cases and an online store for real-time features used during inference. Like Michelangelo, it also is centered around the concept of FGs, the logical groupings of features that often originate from the same data source. FGs can be created by specifying some metadata and inserting a Spark or Pandas dataframe.

Feature reuse is facilitated with the Hopsworks query constructor, which provides a Pandas[2]-like fluent Application Programming Interface (API)[3] for selecting, joining, and filtering features and FGs. The option to specify queries as raw SQL is also provided. A dataset can be created for training by materializing the created query, with the option to specify dataset split sizes (training, testing, validation). Datasets can be exported to external storage for use on other platforms.

An FG can be constructed from external data and existing features in the FS, and a new feature based entirely on reused features from the FS is called a *derived feature*.

[2] `https://pandas.pydata.org/`
[3] `https://www.martinfowler.com/bliki/FluentInterface.html`

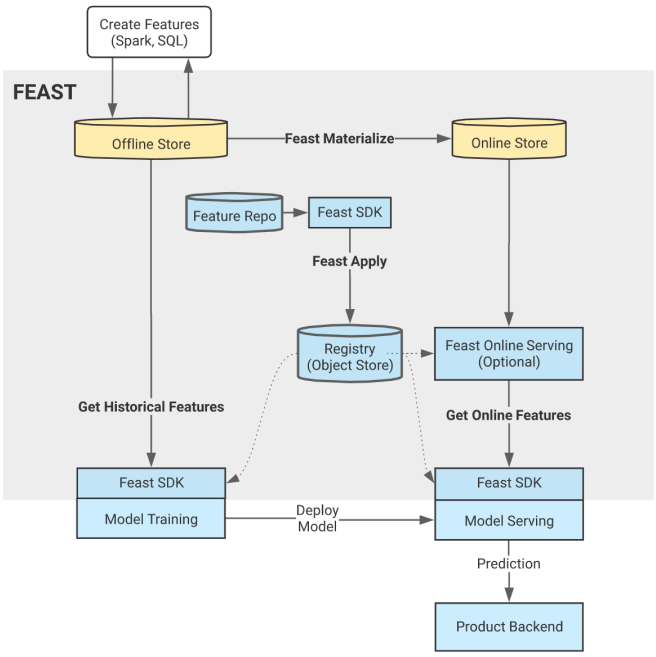


Figure 3.3: Feast architecture. Taken from `https://docs.feast.dev/getting-started/architecture-and-components/overview`.

3.3 Feast

Feast[4] is an open-source FS for ML, similar to the FS module of Hopsworks. It provides a platform for centralized feature definitions, storage, and access, for both training and serving. As seen in Figure 3.3, Feast uses a similar dual-database architecture as Michelangelo and Hopsworks [14].

The Feast Feature Store is configured declaratively as code in a repository, consisting of a YAML file for configuring infrastructure and Python files with feature definitions. All feature definitions and related metadata are cataloged in the feature registry component, making features searchable and discoverable. Feast stores all features as historic time-series values in the offline store, allowing the reproduction of feature values from any point in time. The offline store is used to build training datasets and load the latest feature values into the online store for low-latency serving.

Each project in Feast has a set of *feature views*, an extension of the FG concept found in Michelangelo and Hopsworks. A feature view references a data source, such as a database table or an Apache Parquet[5] file, containing the raw data of which features are made. It may relate to one or more entities, which are a collection of semantically related features, such as a *user* entity or an *order* entity, which

[4] `https://feast.dev/`

[5] `https://parquet.apache.org/`

are intended to be reused across different feature views. Finally, each feature view also has one or more feature definition, which is essentially just a schema specifying the name and data type of the feature. Together with the data source, the feature definitions indicate to Feast where to find a specific feature value. For example, a data source may be given as a specific external database table whence the raw data is retrieved, while the feature is a column of the table.

An ML model consumes features from Feast through a *feature service*, which the user creates per model and model version. The feature service is a grouping of features from one or more feature views and is used when generating training datasets, batch scoring models, and for real-time inference.

3.4 Federated Feature Engineering

Fang *et al.* [42] present a framework for automated and privacy-preserving multi-party FE, the FLFE. The framework addresses the vertical data partitioning variant of the problem, i.e., clients share samples but have different features, as in the case of VFL. Using a structure similar to FL, a central parameter server with a deep neural network classifier determines which feature transformations are useful based on *quantile sketch arrays* [43] uploaded by the participants.

While the FLFE addresses FE in an FL context, it does not contribute to feature reuse. Rather, it represents a complementary component to feature reuse, enhancing the utility of a feature reuse system by potentially being able to create better features securely.

3.5 Limitations of Related Work

Current solutions for ML feature reuse are designed for data that is centrally accessible, which conflicts with FL's fundamental principle of data being private and partitioned among client devices. FL raises the following challenges that are not addressed by existing solutions:

- Data is distributed among participants, while model training is centrally coordinated. While data is local with participants, features should still conform to the expectations of the centrally defined model.
- Participating devices may be significantly resource-constrained and unreliable.
- Clients in cross-device FL cannot be indexed or accessed directly.
- Data volume for an FL application may be locally small, but enormous when aggregated over participants. Combined with the previous item, it means that though it is still desirable to reuse features, a dual-database architecture may be both unnecessary for adequate performance and infeasible due to the additional overhead of having more than one database.
- Participating devices may be heterogeneous in both software and hardware.
- Communications costs are high and networks are unreliable.

As a result, none of the related work can be directly applied to achieve feature reuse for FL. This thesis addresses the first challenge: separating the storage of feature definitions and feature data. Because of time constraints, the scope is limited to focusing on the HFL use case.

Chapter 4

Research Design

This chapter will first present the research motivation and RQ of this thesis, followed by a description of the methodology used to answer the RQ.

4.1 Problem Identification and Motivation

As described in Section 2.1, FL is still an emerging field of research with few real-world applications. However, given the rise of edge computing and increased privacy concerns, it may not be far-fetched to assume that FL will become an important privacy-preserving ML technique. With large engineering organizations benefiting from infrastructure for feature reuse in centralized ML, as described in Section 2.3, the possibility of enabling feature reuse in FL environments should be explored. Feature reuse could, among other benefits, decrease client resource consumption, which is an essential step towards greener IT, as described in Section 2.4. Related work on reusing ML features is based on using centralized FSs, which do not fit the requirements of FL, as described in Section 3.5. This limitation of existing FSs motivates the RQ of this thesis: **How can we reuse ML features across different applications in an FL environment to increase resource efficiency on client devices?**

4.2 Research Method

As the thesis relies on an objectivist/positivist worldview and has an IT system as its instantiated artifact, the Design Science Research Methodology (DSRM) [44] is used as the fundamental methodological framework. Additionally, the general design science guidelines of Hevner *et al.* [45] and evaluation guidelines of Venable *et al.* [46] are used as supplementary methodology resources.

Peffers *et al.* [44] describe the DSRM as consisting of the following six stages:

1. Problem identification and motivation
2. Define the objectives for a solution
3. Design and development

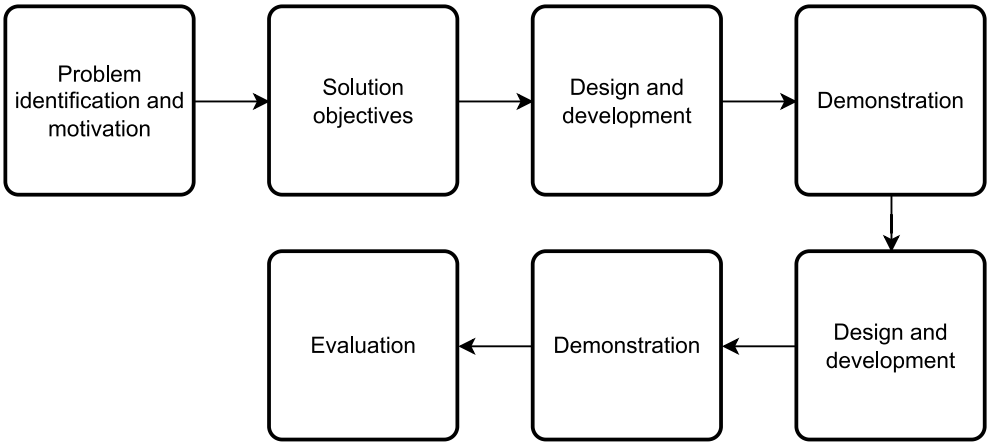


Figure 4.1: Overview of the research process

4. Demonstration
5. Evaluation
6. Communication

The six stages are not meant to be followed sequentially, but can be iterated over in almost any order.

The first activity of the DSRM is *problem identification and motivation*, where the research problem and solution value is justified. This step is covered in Section 4.1.

The second activity of the DSRM is to *define the objectives for a solution* based on the problem definition. Given the problem and RQ stated in Section 4.1, the main objective for a solution is to demonstrate the essential idea of an FS for FL and that it can provide benefits in reducing client resource consumption. Due to the time constraints of the project, the thesis focus is limited to achieving this objective for structured datasets, but this should not lead to a loss of generality for other classes of data such as images and text.

In the *design and development* stage, the artifact is designed and built. The system design comprises the system structure, components and interactions. The design and concepts of the artifact will be inspired by the existing work on reusing ML features described in Chapter 3 and will be adapted to the structural constraints of FL described in Section 2.1. Development is the process of instantiating the design into a working system, the *instantiated artifact*. It can help researchers learn about the drawbacks and advantages of the design decisions and concepts used, which adds to the body of knowledge and can inform later redesigns of the system [47]. A prototype is often built in systems development research to demonstrate the feasibility of the design and the usability of the system's functionalities. However, developing a system from a prototype to a full product and transferring it into an organization is necessary to test the system in a real-world setting. Building the system is also required to conduct empirical studies of the essential ideas of the system.

In the *demonstration* stage, it should be demonstrated that the instantiated artifact can solve one or more problem instances, possibly through experiments, simulations, etc. [44].

The artifact implementation in this thesis is carried out in two iterations of design and development and demonstration, followed by an evaluation. An overview of the order of the research process stages can be seen in Figure 4.1.

The first iteration focused on implementing the bare-bones client-server architecture. Related work was reviewed to understand the design concepts and principles typically used in FSs. A conceptual data model was then created to represent the different types of data in the system. Initial versions of database schemas and APIs were designed, and a first version of the system was developed. The demonstration stage was carried out by reusing features in a simulated FL environment on the *Wisconsin Breast Cancer* dataset[1]. This artifact version successfully demonstrated that features and datasets could be defined on the server, and local data can be stored and used on the client according to the definitions from the server.

The next iteration focused on enabling feature reuse across different applications and adapting the system to being run in a simulated FL environment. The system demonstrated the ability to reuse features in two parallel applications of FL on the *Wisconsin Breast Cancer* dataset.

Following the two design-and-development iterations, the system was evaluated as described in the following section.

4.3 Evaluation Design

In the evaluation stage, the contribution of an artifact to solving the stated problem is observed and measured [44]. The evaluation in this thesis is guided by [46], which provides the Framework for Evaluation in Design Science (FEDS) and a four-step method for constructing the specific evaluation process.

Venable *et al.* [46] present evaluation strategies as paths in a plane, with the first axis being the functional purpose of the evaluation and the second axis being the paradigm of the evaluation study (see Figure 4.2). The functional purpose axis ranges from formative to summative, while the paradigm axis ranges from artificial to naturalistic. All proposed evaluation strategies start out with more formative and artificial evaluation episodes and end with a more summative and naturalistic evaluation episode, but the strategies differ in the number of evaluation episodes and their trajectory along the axes.

4.3.1 Step 1: Explicating Goals

The first step of the FEDS process is to make explicit the goals of evaluation. Venable *et al.* [46] list four possibly competing goals of an evaluation project: *Rigor*, *uncertainty and risk reduction*, *ethics*, and *efficiency*.

[1] `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`

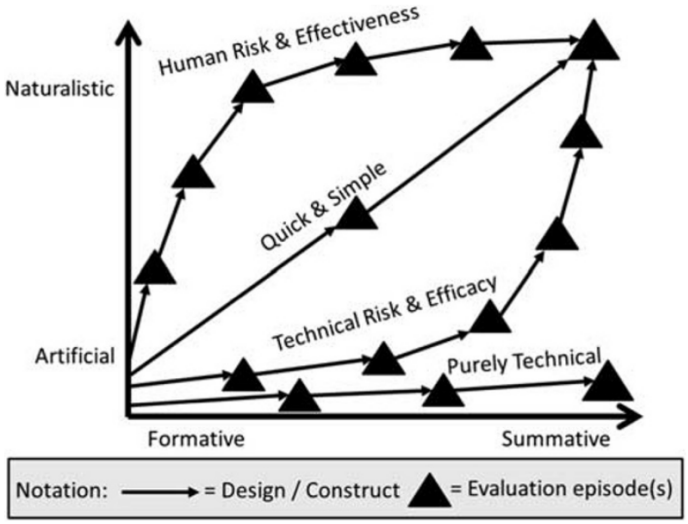


Figure 4.2: Illustration of the Framework for Evaluation in Design Science (FEDS) framework with the different proposed evaluation strategies. Taken from [46].

According to Venable *et al.* [46], there are two aspects to rigor in DSR: efficacy and effectiveness. Effectiveness refers to the rigor in establishing that the artifact works in a real-world setting, while efficacy is the rigor in establishing cause/effect while eliminating confounding variables. While the *effectiveness* aspect of rigor is important, it requires observation of system use in a real-world scenario, which is difficult in this project (in part due to FL still being an emerging technology). For this reason, the evaluation will have a lower emphasis on rigor.

The goal of uncertainty and risk reduction is concerned with reducing risk and uncertainty relating to human social/use risks and technical risks [46]. An example of a human risk is the artifact not working well in the social setting, while a technical risk would be that technological limitations make it impossible to get the artifact working. As FL is an emerging field of research with relatively few real-world applications reported so far [48], and the literature on FSs being even more scarce, it follows that research combining FL and FSs is affected by a high degree of design uncertainty. Mitigating this uncertainty is an important goal of the evaluation in this thesis.

Ethics relates to addressing risks to people, animals, the environment, etc., both currently and in the future, and is especially important for safety-critical systems [46].

Efficiency relates to resource usage, mainly in terms of time and money [46]. Being a thesis, there is a fixed deadline, a single researcher, and virtually no budget, so resource usage must be highly efficient. Efficiency is thus an important goal in artifact evaluation.

The four evaluation goals listed by Venable *et al.* [46] have the following priorities in this thesis:

1. Efficiency (high)
2. Uncertainty and risk reduction (high)
3. Rigor (moderate)
4. Ethics (low)

4.3.2 Step 2: Choosing the Evaluation Strategy

The second step of the FEDS process it to choose a strategy for evaluation. According to Venable *et al.* [46], there are four different evaluation strategies. The *Quick & Simple* strategy is suitable for small and simple designs with low risk and uncertainty relating to technical or social aspects. The *Human Risk & Effectiveness* strategy may be appropriate in any of the following circumstances: 1) The major design risk relates to users or social aspects, 2) performing a real-world evaluation with real users is relatively cheap, 3) it is a critical evaluation goal to establish real-world utility. The *Technical Risk & Efficacy* strategy may be appropriate in any of the following circumstances: 1) The major design risk is technical, 2) real-world evaluation is prohibitively expensive, 3) a critical goal of the evaluation is to establish that the utility of the artifact is not due to something else. The *Purely Technical Artifact* strategy is only appropriate for purely technical artifacts, or if the artifact will only be deployed well into the future.

This thesis employs the *Purely Technical Artifact* evaluation strategy for two reasons: 1) The uncertainties of the project are mainly technical and not social, as the FS concept has previously been successfully applied in multiple large organizations (see Section 2.3), and the novel application to FL is a technical detail. 2) The artifact will not be deployed anytime soon.

4.3.3 Step 3: Determining Properties to Evaluate

The third step of the FEDS process is to determine which properties of the artifact to evaluate. Following the guidelines of Venable *et al.* [46], artifact evaluation properties are determined by first framing potential evaluands, then aligning the candidate evaluands with the goals stated in Section 4.3.1, and, finally, considering the nature of the evaluation strategy chosen in Section 4.3.2.

Based on the RQ from Section 4.1 and stated solution objectives in Section 4.2, at least two candidate evaluands can be inferred: 1) The ability to reuse features across different FL applications, and 2) system resource usage with respect to the level feature reuse. System resource usage metrics such as, CPU, memory, disk and network usage, all of which are common metrics used for evaluation in FL research [48], are relevant potential evaluands. Resource usage is an important issue for FL applications as clients may be limited in hardware capabilities and computations should not affect system performance or stability [19]. Given that FL is commonly applied to sensitive data, security and privacy are also relevant properties to evaluate. Further, as the client devices in FL can be highly heterogeneous [19], production deployment of the artifact would require a high degree of portability and interoperability.

To answer the RQ, the evaluation must necessarily verify that features are reusable across FL applications and is compatible with the constraints of FL. This achieves reduction of uncertainty and risk, and should be quite resource-efficient, thus aligning with the two highest-priority evaluation goals. Additionally, it mitigates technical design risks, which aligns with the evaluation strategy *Purely Technical Artifact* [46]. Further, to verify that feature reuse decreases resource usage on clients, CPU and disk usage are also used as metrics in the evaluation.

4.3.4 Step 4: Designing Individual Evaluation Episodes

The final step in the FEDS process is designing the actual evaluation episodes, which should take into account constraints in the environment and resources [46]. This thesis is constrained in both time (21 weeks), human resources (one master's student) and budget (nigh non-existent). In the interest of evaluation efficiency, only a single evaluation episode is planned.

Evaluation is conducted by the author after the artifact design iterations by simulating use of the artifact in an FL environment, and comprises two parts. The first part of the evaluation consists of two experiments and verifies that the design is compatible with FL and enables feature reuse across different applications. The second part measures CPU and disk usage metrics to verify the claim that feature reuse decreases resource usage.

Federated Learning Compatibility Evaluation

The *Superconductivity*[2] [15, 16] dataset is used in the first experiment for a regression task. It is already cleaned, has a high number of pre-engineered numerical features, making it convenient to use in this experiment. The second experiment uses the *Adult*[3] [15] dataset for binary classification. It requires some cleaning and processing before usage, such as removing rows with missing data, one-hot encoding categorical features, etc. The *Wisconsin Breast Cancer*[4] dataset, while small, is useful for efficiently testing the system between design iterations. The datasets are chosen because they are structured (tabular), represent different types of learning tasks (classification and regression), and the Adult and Superconductivity datasets have previously been used as structured data sources for FL research, as in [49–53] and [54–58]. A summary of the datasets can be found in Table 4.1.

Rather than attempting to find multiple different applications requiring overlapping features, a more contrived approach will be taken to simulate real-world feature reuse. Two instances of the same application (e.g. predicting income over/under USD $50,000 in the case of *Adult*) are run in parallel at four quintiles of feature reuse, ranging from 0% to 60%. For the first part of the evaluation—demonstrating compatibility with FL—it suffices to validate that the system operates as intended during FL tasks. This part is conducted with one CS and five

[2] `https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data`

[3] `https://archive.ics.uci.edu/ml/datasets/adult`

[4] `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)`

Table 4.1: Datasets used in evaluation.

Dataset	Task type	Samples	Features
Superconductivity [15, 16]	Regression	21,263	81
Adult [15]	Classification	48,842	14 (22 after FE)
Wisconsin Breast Cancer [15, 17]	Classification	699	10

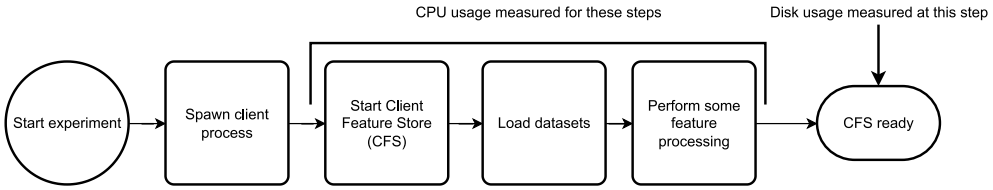


Figure 4.3: CPU and disk usage are measured for the indicated steps on the client device.

clients. The classification tasks (Adult dataset) and regression task (Superconductivity) are run separately, with two models in parallel.

Resource Usage Evaluation

For the second part of the evaluation, total CPU and disk usage of clients is measured during the FE stage. Reduced CPU and disk usage should be observed with increasing levels of feature reuse. Total CPU time for one client is measured for the duration of the client initialization, dataset loading and FE process (see Figure 4.3), and total disk usage (measured by database file size) is measured after feature ingestion at different levels of feature reuse, ranging from 0% to 60%. The Adult, Superconductivity, and Wisconsin Breast Cancer datasets are used concurrently for this experiment.

Definition of Feature Reuse Level

The level of feature reuse is defined as follows: Given two applications A_1, A_2, let F_1 and F_2 be the set of features used by each application, respectively. The level of feature reuse for A_2 with respect to A_1, $\rho(F_1, F_2)$, is then calculated as

$$\rho(F_1, F_2) = \frac{\left|F_1 \bigcap F_2\right|}{\left|F_2\right|} \tag{4.1}$$

The experiments in this thesis use two applications per dataset with identical feature set sizes, i.e., $|F_1| = |F_2|$, so the feature reuse level is effectively symmetrical ($\rho(F_1, F_2) = \rho(F_2, F_1)$).

Chapter 5

Results

This chapter will present the design and implementation of a system for feature reuse in HFL, named the Federated Feature Store (FFS), followed by the experimental setup and evaluation results.

5.1 Design and Implementation of the Federated Feature Store

To achieve feature reuse on FL client devices, feature data needs to be both 1) private and local—to satisfy the FL constraint—and 2) locally centralized on each device—to avoid redundant resource consumption. Feature definitions are managed by the organization deploying the FL application, so it should have control over the single source of truth of features definitions to be used on clients. This suggests a client-server architecture for the system, with a server component managing feature definitions and a client component storing the actual local feature data on each client device. The server component is called the Feature Registry (FR), as it fulfills essentially the same purpose as the Registry component in Feast, and the client component is called the Client Feature Store (CFS), as this is where the local feature data is stored on each client device. The overall system architecture is illustrated in Figure 5.1.

As multiple features are often naturally grouped together for convenience of computation and/or consumption, the FG will be used as the atomic unit of feature storage, following the convention of previous FSs (see Chapter 3). This means that all features will be defined as part of an FG (though, if necessary, it could be a singleton group with only one feature). The feature data used for a particular FL application can then be constructed by combining the FGs containing all the necessary features. Consuming feature data, which may consist of features from many different FGs, should be straightforward for FL applications, and the set of FGs in use for an application, as well as their method of combination, should be well-defined. This means that the knowledge of how to construct and query a dataset should not lie with the FL application, but should be the responsibility

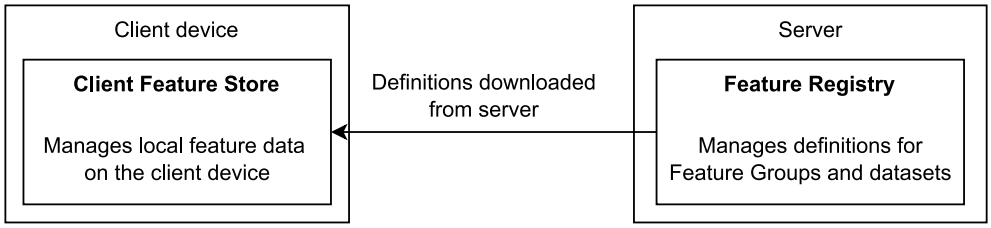
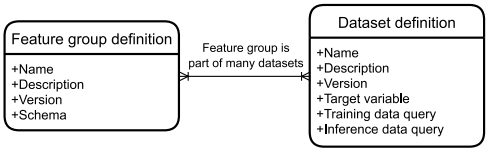


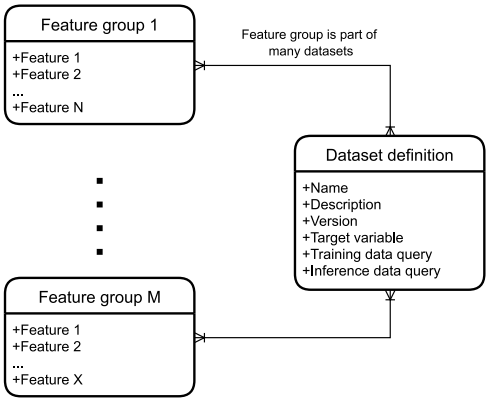
Figure 5.1: Architectural overview of the two main system components in the Federated Feature Store (FFS): the Client Feature Store (CFS) and the Feature Registry (FR). The FR hosts and manages definitions for datasets and Feature Groups (FGs) on a server. The CFS runs on each client device participating in Federated Learning (FL) and manages local feature data.

of the FFS. For these reasons, datasets should also be defined alongside FGs in the FFS. Feature reuse across different applications is then achieved by using an existing FG in the definition of a new dataset.

As the FR is responsible for holding the single source of truth for definitions, it stores definitions for both FGs and datasets. To facilitate feature reuse, it is not enough for FG definitions to only dictate the structure and data types of their component features (the *schema*); there should be some descriptive metadata alongside it to further document the features and help feature consumers understand if a given FG is suitable for their specific application needs. The design in this thesis specifies the basic metadata fields *Name*, which is used to identify the FG, *Version*, to enable FG definition versioning, and a *Description*, which may provide details such as semantics and business rules of the component features. A dataset definition needs to specify how the FGs should be combined to construct the dataset on the client, and should also document intended use and pertinent details, be versionable, and be addressable by a human-friendly name. Dataset definitions therefore have the metadata fields *Name*, *Version*, *Description*, and *Training data query*. For convenience, the *Target variable* of a dataset is also specified, as the FFS then can return the input variables of the dataset separately from the target variable when retrieving training data samples. There should also be some relation between FG definitions and dataset definitions so that the necessary FG definitions can be found when a given dataset definition is requested. As a dataset may be constructed from many FGs and an FG may be used in many datasets, this relation is many-to-many. For the sake of simplicity, the many-to-many relation between datasets and FGs in the CFS is not present in the artifact implementation, as the queries specified in the dataset definitions exclusively query the FG tables, so the relation is not actually needed for querying the data. However, the relation would likely be desirable in a production implementation, for example to ensure data integrity or to easily determine which FGs are in use by which (if any) datasets. Figure 5.2 illustrates the conceptual data models for datasets, FGs, and their respective definitions. Figure 5.2a illustrates the data model for the FR, where a dataset may combine many FGs, and, crucially for feature reuse, an FG



(a) Data model for the Feature Registry (FR) illustrating the relationship between datasets and Feature Groups (FGs). One dataset may use multiple FGs, and an FG may be reused across many different datasets.



(b) Data model for the Client Feature Store (CFS). The dataset entity mirrors the one for the Feature Registry (FR), while the entities for Feature Groups (FGs) are dynamically created based on the schema found in the FG definition. One dataset may use multiple FGs, and an FG may be reused across many different datasets.

Figure 5.2: Conceptual data models for the Feature Registry (FR) and Client Feature Store (CFS).

may be used in many different datasets. Figure 5.2b shows the CFS data model, in which each FG is a separate entity with a many-to-many relation with the dataset definitions entity.

In the multitenant FL scenario (i.e., multiple different FL applications running on the same client device), all the FLCs retrieve their training data from the CFS, as illustrated in Figure 5.3. During local inference, the trained models may also use data from the CFS to make predictions. The FR and all CSs are depicted as residing on the same server for the sake of simplicity, but this is by no means necessary. It is entirely possible to have both the FR and each of the CSs residing on different servers.

5.1.1 Feature Registry Design

The FR is responsible for creating, storing and serving definitions of FGs and datasets to CFSs, fulfilling roughly the same role as the *Registry* component of Feast described in Section 3.3. Users should be able to create new dataset and FG definitions, and CFSs should be able to download definitions to the client devices

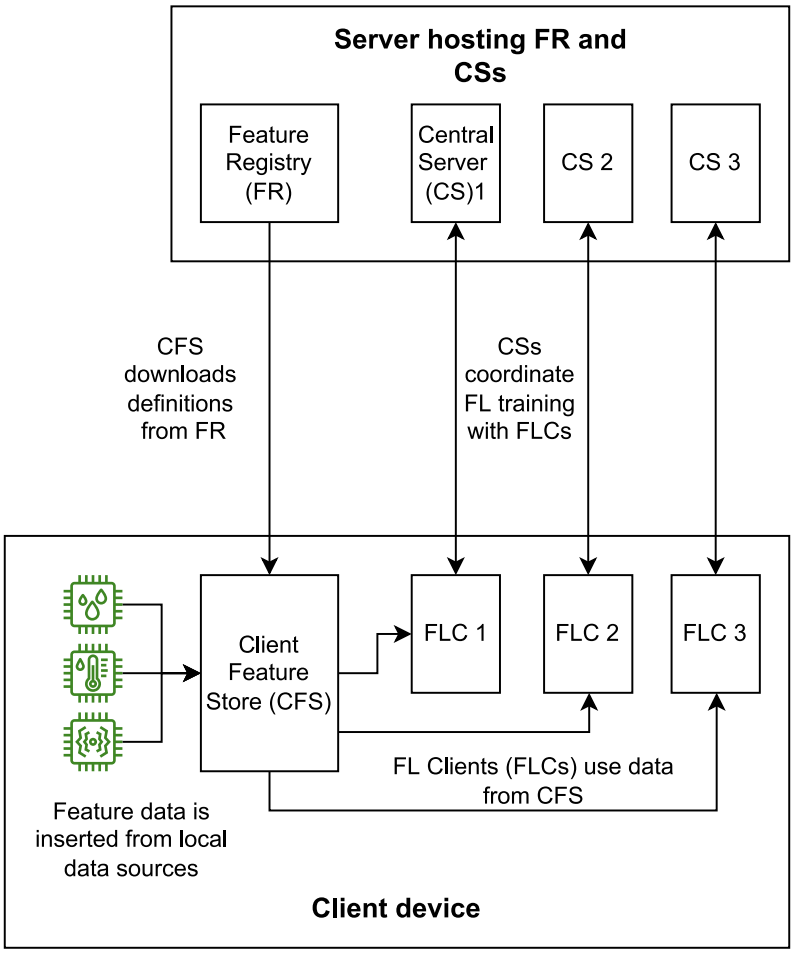


Figure 5.3: Use of the Federated Feature Store (FFS) in a multitenant Federated Learning (FL) scenario. Each client device may have multiple Federated Learning Clients (FLCs), which all retrieve feature data from a single Client Feature Store (CFS). While FL generally involves many clients, only a single client device is depicted for the sake of simplicity.

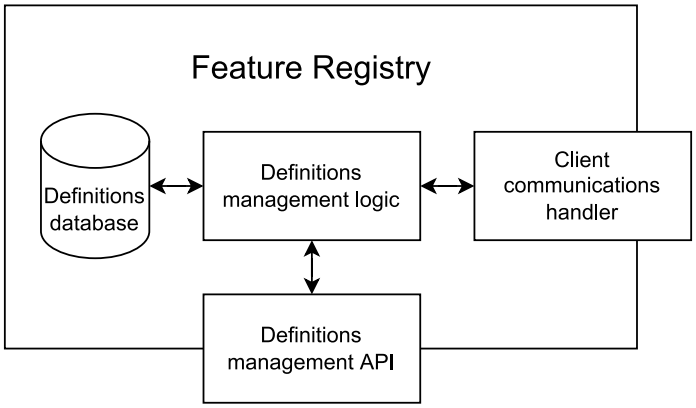


Figure 5.4: Feature Registry (FR) design.

on which they reside. Thus, the FR must have a storage mechanism, network communications capability, and a definitions management interface and corresponding logic, illustrated in Figure 5.4.

FG and dataset definitions are structured (tabular) and relational in nature, so a relational database is used for definitions storage in the FFS implementation The definitions management logic ties together the database, the CFS communications and the user-facing API, and is responsible for executing the processes described in Section 5.1.2. The database schemas for the FR are based on the data model seen in Figure 5.2a, and can be found in Appendix B.

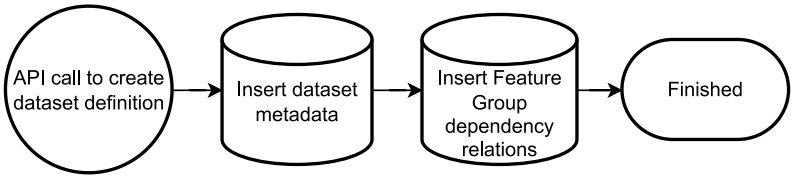
5.1.2 Feature Registry Processes

The FR must be able to execute three different processes to fulfill its role as a centralized definitions management component: 1) create FG definitions, 2) create dataset definitions, and 3) respond to dataset definition requests received from CFSs.

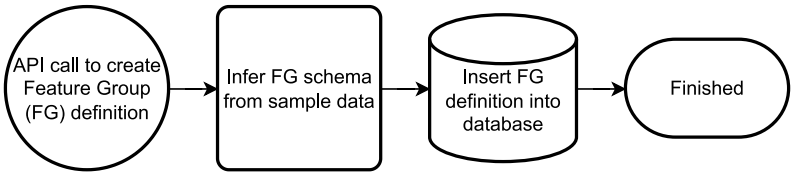
An FG definition is created through the FR API from user-supplied FG metadata (name, description) and sample feature data. The FG schema (columns and data types) is automatically inferred from the sample data for the sake of implementation convenience. FG metadata and schema are then inserted into the FR database and available for download by CFSs, as seen in Figure 5.5a.

Dataset definitions are created through the FR API with user-supplied metadata (name, description, target variable), names of required FGs, and queries for retrieving training data. The dataset metadata and queries are inserted into the FR database, followed by the creation of the FG dependency relations in the FG-dataset junction table (see Appendix B). The process is illustrated in Figure 5.5b.

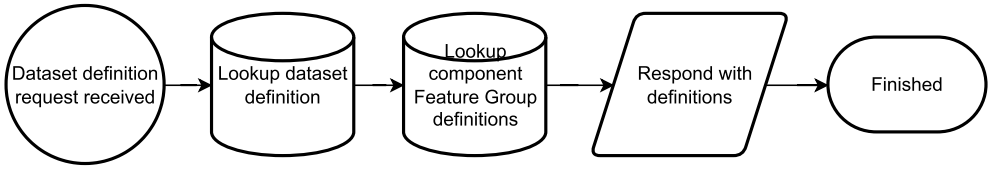
Upon receiving a request for a dataset definition from a CFS, the FR does a database lookup for the requested dataset definition based on the name and version specified in the request. The FG dependencies of the dataset are then retrieved by performing an `INNER JOIN` between FG definitions table and the FG-



(a) Feature Group (FG) definition creation process.



(b) Dataset definition creation process.



(c) Dataset definition request handling.

Figure 5.5: Flowcharts illustrating processes of the Feature Registry (FR).

dataset junction table (see Appendix B for further details on the FR tables). Finally, the FR responds to the CFS request with the dataset and FG definitions, as seen in Figure 5.5c.

5.1.3 Client Feature Store Design

The CFS is responsible for ingesting and storing local feature data on the client devices following the FG definitions from the FR and providing FLCs access to the data for use during training and inference. Thus, the client must have a storage mechanism, a communications component for downloading definitions, an API ingesting local feature data, and an API providing access to the local feature data. The high-level CFS design can be found in Figure 5.6.

A relational database is used for the purpose of data storage, despite no entity relations being explicitly modeled in the CFS (see explanation earlier in Section 5.1). This is to have the convenience of specifying and storing queries as SQL, and because the thesis is focused on use cases with structured data, for which relational databases are well-suited as a storage mechanism.

The communications component is responsible for sending dataset definition requests to the FR and relaying the response back to the management logic.

It should be easy for FLCs to retrieve a new batch of training data in each round of FL training, so it is desirable to make the CFS responsible for maintaining the state of iteration over the local feature data between each round of FL training. The Dataset Instance (DI) component fulfills this purpose, representing a usage

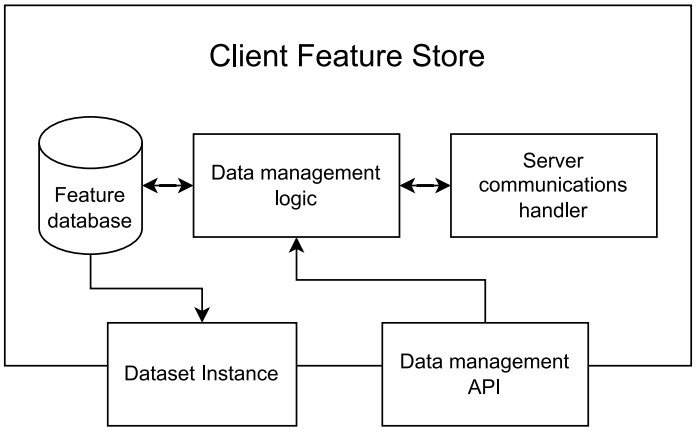


Figure 5.6: Client Feature Store (CFS) design.

instance of a given dataset, similar to a dataset in Hopsworks or a feature service in Feast.

Finally, the management logic is responsible for the overall execution of most CFS processes described in Section 5.1.4, such as determining dataset definitions that need to be downloaded, creating FG tables, handling feature data insertion, and providing DIs.

5.1.4 Client Feature Store Processes

To satisfy the responsibilities outlined in Section 5.1.3, the CFS should have processes for downloading required datasets, ingesting feature data, retrieving training data, and retrieving inference data.

At CFS startup, the FR address and required datasets for all applications on the client are specified by the user. The CFS queries its internal database to determine if any required dataset definitions are missing. For each missing dataset definition, a request is sent to the FR, which responds with the dataset definition and all its FG dependency definitions. Downloaded dataset definitions are inserted into the CFS database, while a new table is created for each new FG definition. The CFS is then ready for operation. See Figure 5.7a for an illustration.

Before any data can be retrieved from the CFS, the local feature data must first be inserted. Using the CFS API, the user specifies the FG name and provides the feature data, which is then written to the correct FG database table (Figure 5.7b).

As the FL training process begins, the FLC requests the relevant dataset by name from the CFS API. A DI is returned, which can be used to retrieve training batches of feature data from the database (Figure 5.7c). The reason for not directly retrieving data through the CFS is to simplify the logic that maintains iteration state between FL training rounds.

When the FLC is selected for a training round by the CS, it requests a batch of training data from the DI, which executes the training data query from the dataset definition to join FGs and create the resulting training data samples (Figure 5.7d).

The DI operates lazily, so the query is only executed on the first batch request, while subsequent requests are handled by simply iterating over the database cursor and returning batches.

In the inference stage of operations, the client requests feature data from the CFS, which queries the database with the inference data query specified in the dataset definition (Figure 5.7e).

5.1.5 User Interactions With the Federated Feature Store

Figure 5.8 provides a flowchart describing user interactions with the system when deploying a new FL task. Users of the FFS will interact with the system on both the server-side and client-side. When a new FL application is going to be deployed, a dataset definition for the particular task should be created. This is done on the server, where the FR resides, managing all definitions. The dataset is defined by specifying metadata (name, description, etc.), FG dependencies (FGs necessary to construct the dataset), and queries for retrieving training data. Existing FGs should first be searched by the user to discover any potentially reusable features, while any new features should be defined with metadata (name, description, etc.) and their schema (columns and data types). This concludes user interactions on the server.

On the client devices, the required dataset(s, if multitenant) is specified before CFS initialization. If the dataset (or alternatively, all the FGs in the dataset) was already in use, the CFS is already populated with the necessary data and the client is ready to proceed with the training, otherwise local feature data should be inserted first. To use a dataset for a given application, a DI is obtained from the CFS, which provides access to batches of training data samples for the FLC.

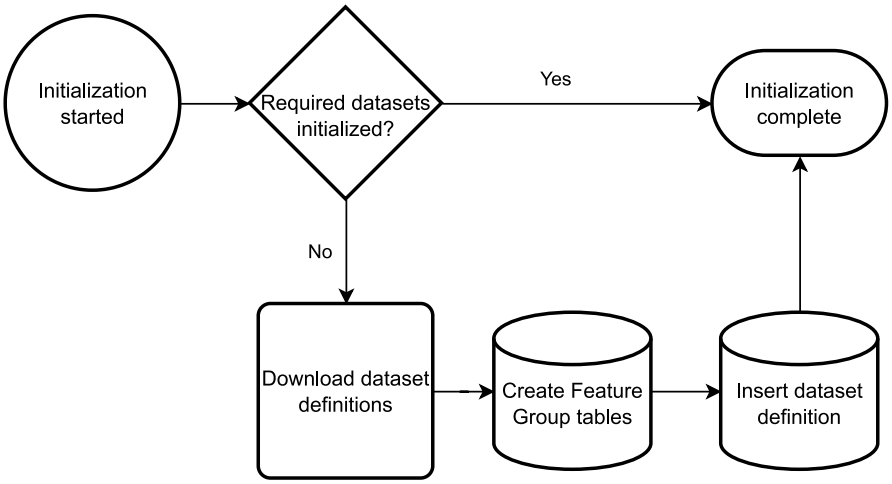
5.1.6 Implementation Details

The entire FFS is implemented in Python for the sake of easy compatibility with ML and FL frameworks. The CFS and FR each have a SQLite database for storing data and definitions, respectively. Client-server communications between the components is implemented over RPC using gRPC.

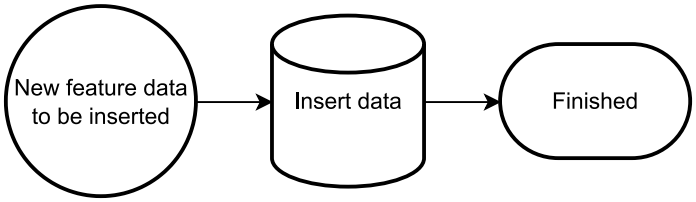
Pandas DataFrames are used in all data handling in the FFS. In the CFS, they are used for ingesting and returning feature data. In the FR, sample data for new FG definitions is provided by the user in a Pandas DataFrame. The FG schema can then be automatically inferred using the `pandas.io.sql.get_schema()`, which generates the `CREATE TABLE` statement for the FG table.

Only a few SQL statements are manually written, namely the `CREATE TABLE` statements for the dataset definitions table on the CFS, and the `CREATE TABLE` statements for the dataset definitions, FG definitions and dataset-FG junction table on the FR. All other SQL statements are generated via the PyPika library, (or Pandas, in the case of schema generation, as described in the previous paragraph).

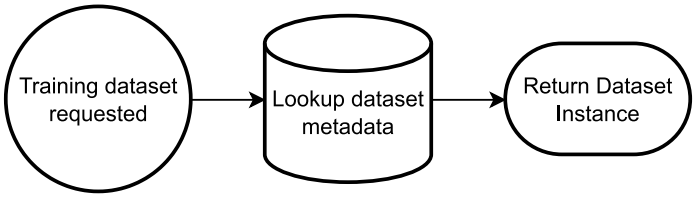
An overview of the packages used to implement the FFS can be found in Table 5.1.



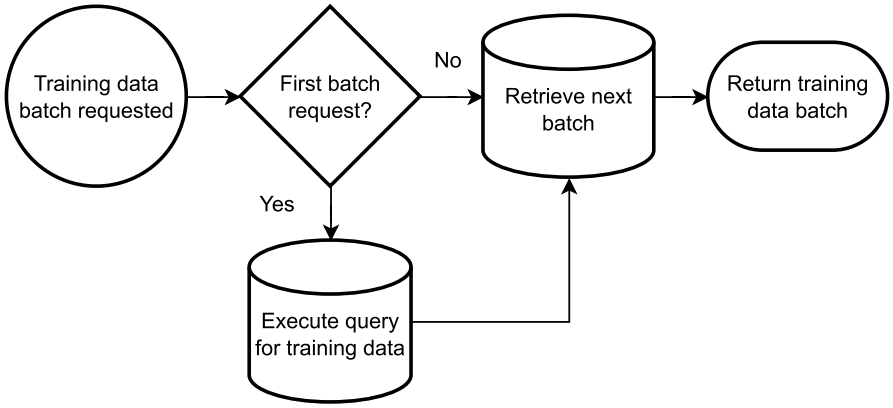
(a) The CFS initializes by downloading any missing dataset definitions and creating missing FG tables.



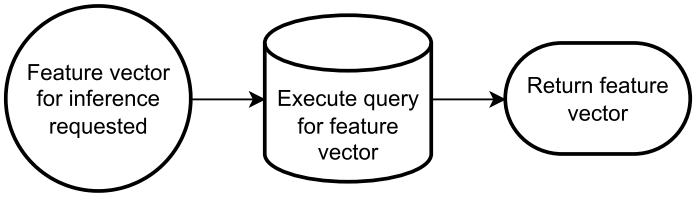
(b) Feature data insertion.



(c) Request for dataset.



(d) Request for a batch of training data. The query is only executed on the first call, subsequent calls iterate over the database cursor.



(e) Request for inference data.

Figure 5.7: Flowcharts illustrating the processes on the Client Feature Store (CFS).

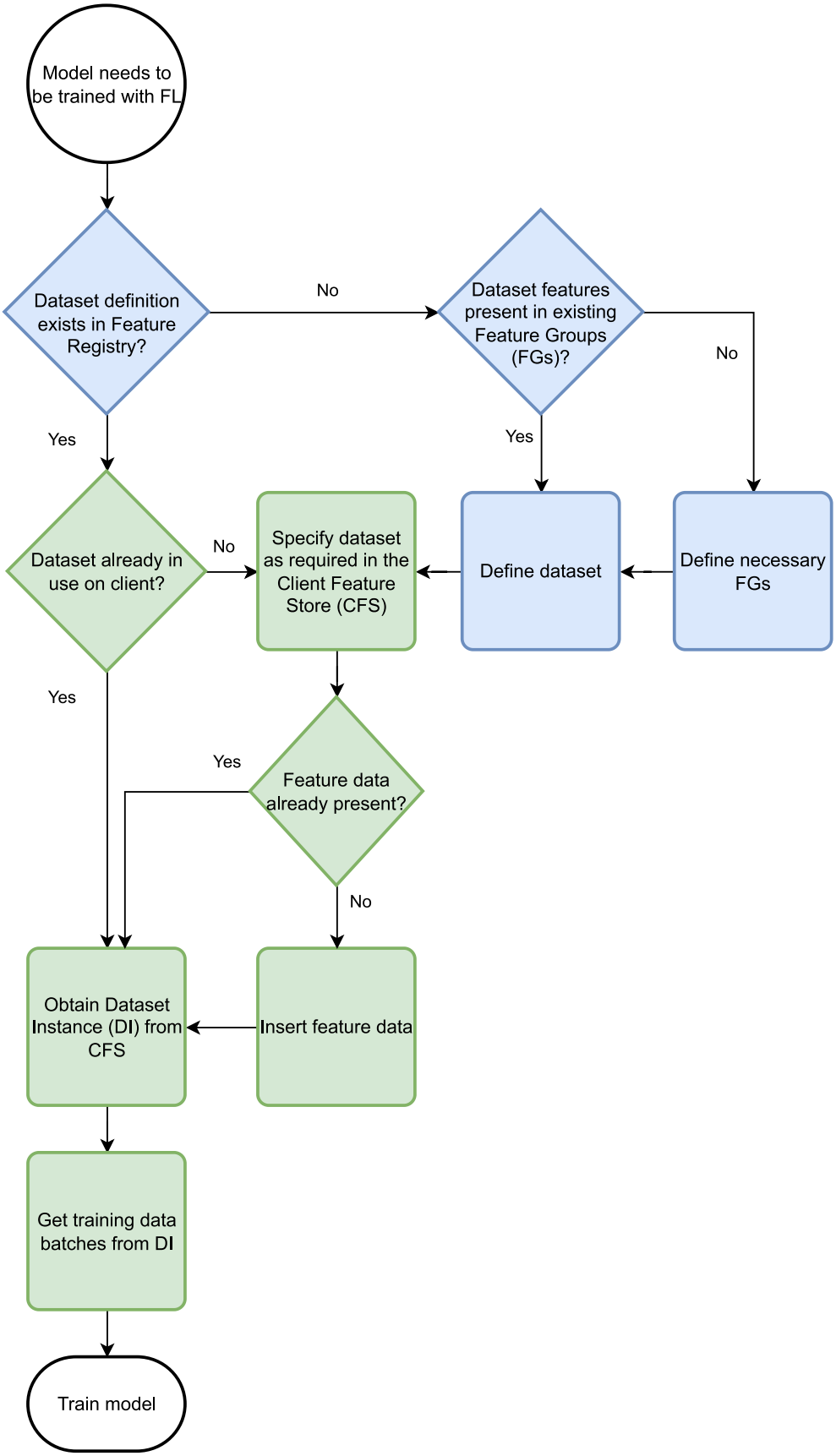


Figure 5.8: Flowchart illustrating usage of the Federated Feature Store (FFS). Green shapes are interactions on the client device, while blue shapes indicate interactions on the server.

Table 5.1: Packages used to implement the Federated Feature Store (FFS).

Package	Version	Use
Python[1]	3.10	Implementation language for whole system
SQLite[2]	3.38.5	Storage in both FR and CFS
sqlite3[3]	2.6.0	Python SQLite API, used to access the SQLite databases
grpcio[4]	1.43	Provides RPC communications between FR and CFS
grpcio-tools[5]	1.43	Code-generation for the RPC interfaces
PyPika[6]	0.48.9	SQL query builder on FR and CFS
Pandas[7]	1.4.1	DataFrames used for all data interactions and for SQL schema generation

Table 5.2: Software used in the experimental environment.

Software	Version	Use
Arch Linux[8]	5.17.6	Operating system
Python	3.10	Implementation language of experimental environment
Scikit-learn[9] [59]	1.0.2	ML models
Flower[10] [60]	0.18	FL simulation

5.2 Experimental Setup

5.2.1 Environment

Experiments are run on a single desktop machine with a 4 GHz AMD FX8350, 16 GB DDR3 RAM at 1600 MHz, running Arch Linux with kernel 5.17.6. Python is used to build the environment in which the system is tested. The Flower framework is used to simulate an FL environment with the `FedAvg` model update aggregation strategy. The `SGDRegressor` and `LogisticRegression` Scikit-learn models were used for regression and classification, respectively.

An overview of the software used in the experimental environment can be seen in Table 5.2.

5.2.2 Creating Synthetic Feature Groups for Experiments

To evaluate the system at different levels of feature reuse, as planned in Section 4.3.4, a method is needed to simulate different levels of feature reuse for a given dataset. As most available structured datasets come in the form of a single denormalized table, it is most convenient to create FGs synthetically by splitting up the datasets into different tables with roughly equal numbers of feature columns. A primary key column is added to reconstruct each training sample (i.e.,

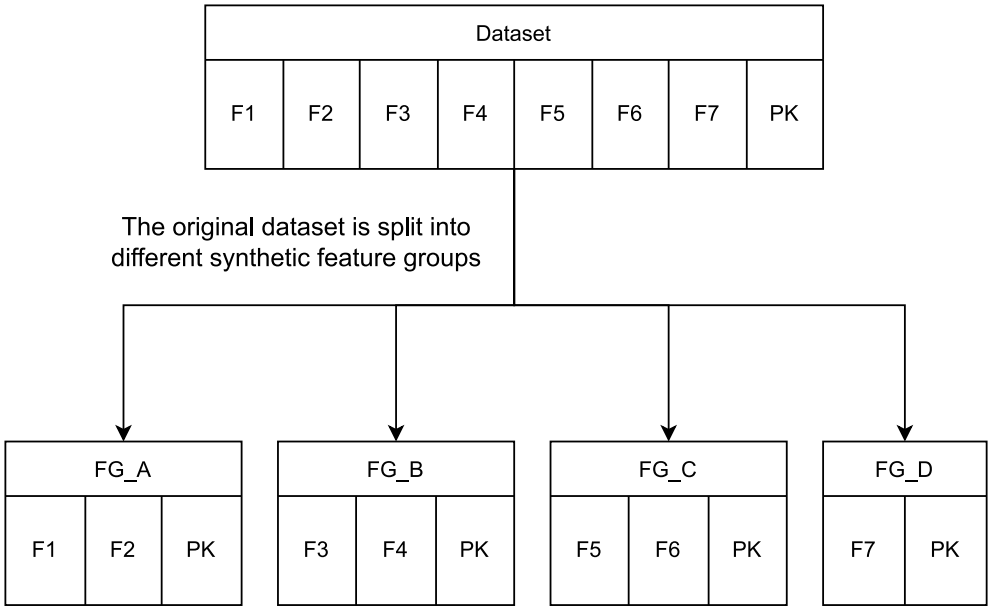


Figure 5.9: Illustration of how synthetic Feature Groups (FGs) are created from a dataset. The original dataset with features F1–F7 and primary key PK is split into FGs FG_A–FG_D.

a row in the dataset). The grouping process is entirely deterministic, grouping features by their column adjacency in the original dataset file. The resulting FGs are dubbed *synthetic* because it would not be natural in a real-world scenario to split the dataset and join it again shortly after. See Figure 5.9 for an illustration of the creation process for synthetic FGs. To retrieve training samples from the created FGs, an SQL query is specified in the dataset definition as a simple inner join between all the FG tables of the dataset on the primary key column.

5.2.3 Experimental Feature Group Reuse

FG reuse at four different quintiles, 0%–60%, is simulated for each experiment. All datasets are split into five different FGs, as described in Section 5.2.2. FGs are reused based on the order of their features in the original dataset. At 0% reuse, all FGs are stored twice, and the models use disjoint sets of FGs. At 60% reuse, 40% of the FGs are stored redundantly, while both $m1$ and $m2$ use the remaining 60%. See Equation (4.1) for the mathematical definition of feature reuse level in this thesis.

Example: For a dataset with feature columns A, B, C, D, E, F, G, H, I, and J, the features are grouped AB, CD, EF, GH, and IJ. At 0% reuse, the database will contain FGs AB1–IJ1 used by $m1$ and AB2–IJ2 used by $m2$. At 20% reuse, the FG AB will be reused for both FL models while CD–IJ are duplicated. Duplication means that the CFS database will contain one table for AB and two tables for each of the FGs CD–IJ, such that the database contains the following tables: AB (used

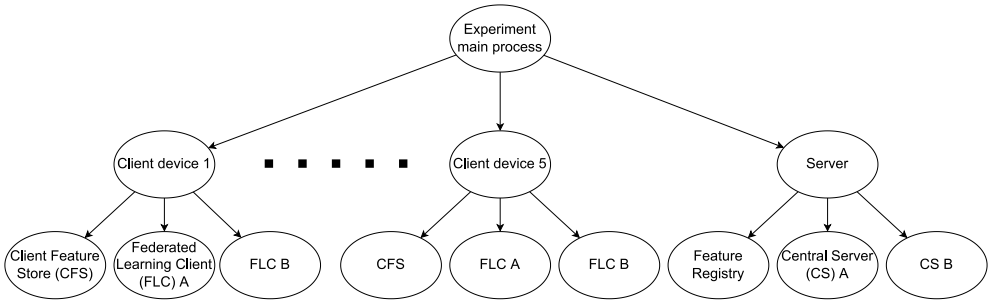


Figure 5.10: Process tree for compatibility evaluation. Client devices 2–4 are omitted for brevity. Each client device has two FLCs training models concurrently on applications A and B, coordinated by Central Server (CS) A and B running as child processes of the server process. A and B are based on the same dataset, with varying levels of feature reuse between, as described in Section 5.2.3.

by both $m1$ and $m2$), CD1–IJ1 (used only by $m1$), CD2–IJ2 (used only by $m2$).

5.2.4 Federated Learning Compatibility Evaluation

The system is used in two experiments to evaluate its compatibility with FL and to demonstrate that the system can operate with a wide range of feature reuse levels in concurrent applications. The specific regression and classification models used are chosen for the sake of simplicity in integration with the FL simulation framework.

Each experiment trains two identical models, $m1$ and $m2$, in parallel on five different client devices. One hundred rounds of FL training are performed per experiment repetition. Each client has a single CFS, from which both local models retrieve training data. 80% of the training data is divided equally among the five clients, while the remaining 20% is used for model evaluation by the CS post-training. The FL training process is repeated ten times for each level of feature reuse. Figure 5.10 illustrates the experiment with a process tree.

Superconductivity Dataset Experiment

This experiment uses the *Superconductivity* dataset[11], with 81 features and 21,263 samples, for multivariate regression. Each sample is a feature-engineered data point of a superconductor, with the critical temperature of the material being the target variable to predict. An ID column is added as the primary key for the dataset. The `SGDRegressor` model from Scikit-learn is used with the parameters *max_iter=1* (we only want a single iteration per training round of FL) and *warm_start=True* (to start with the weights received from the CS).

[11] `https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data`

Adult Dataset Experiment

This experiment uses the *Adult* dataset[12], which has 14 features and 48,842 samples. The dataset is used for binary classification of adult income based on US census data, where each sample is labeled either >50K or <=50K. The dataset is cleaned[13] and categorical features are one-hot encoded during the FE process. The `LogisticRegression` classifier from Scikit-learn is used with the parameters `penalty="l2"`, `max_iter=1`, and `warm_start=True` as suggested in the Flower quickstart guide[14].

5.2.5 Evaluation of Resource Usage

To verify that increased feature reuse results in lowered resource consumption during FE and feature ingestion, CPU and disk usage are measured during and after FE and feature ingestion, respectively. The resource usage experiment is repeated ten times per level of feature reuse (described in Section 5.2.3) for all four levels. The *Adult*, *Superconductivity*, and *Wisconsin Breast Cancer* datasets are used concurrently for this experiment.

CPU Usage During Feature Engineering and Ingestion

CPU usage is measured for a single client at varying levels of feature reuse from the client process spawns until the FS is populated with data, involving the steps of data loading, data processing/FE, and data ingestion, as indicated in Figure 4.3.

CPU usage is measured with the `getrusage()` function from the Python `resource` package, which uses the platform implementation of the `getrusage()` POSIX system call. The function is called in the client process with the *RUSAGE_SELF* flag. The function returns cumulative CPU time in user and system mode, among other platform-dependent statistics. Only the user time is considered for this experiment, but system CPU usage can also be found in Table 5.3 under the column STIME. The process tree in Figure 5.11 illustrates for which processes CPU usage is measured.

Disk Usage After Feature Engineering and Ingestion

Disk usage for a single client is measured at varying levels of feature reuse after FE and feature ingestion are completed. The file size of the CFS SQLite database is used as the metric for client disk usage and is measured with the `du` command using the `-k` flag to get a 1K block size. The database file is deleted before each repetition to ensure identical circumstances in each experiment iteration.

[12] `https://archive.ics.uci.edu/ml/datasets/Adult`

[13] Data cleaning follows the steps from `https://ryanwingate.com/projects/machine-learning-data-prep/adult/adult-cleaning/`.

[14] `https://flower.dev/docs/quickstart-scikitlearn.html`

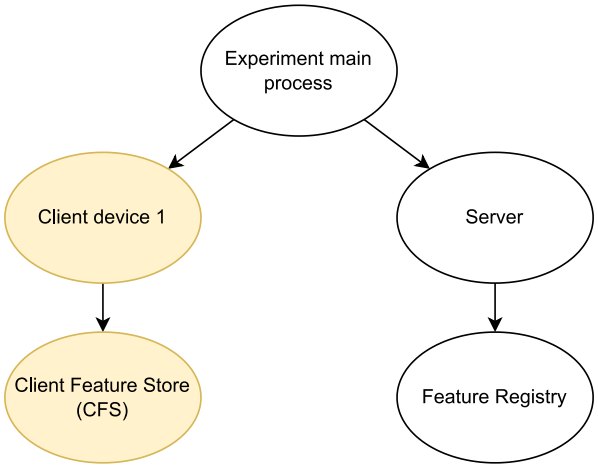


Figure 5.11: Process tree during the resource usage experiment. Yellow ellipses indicate for which processes the CPU usage is measured.

5.3 Experimental Results

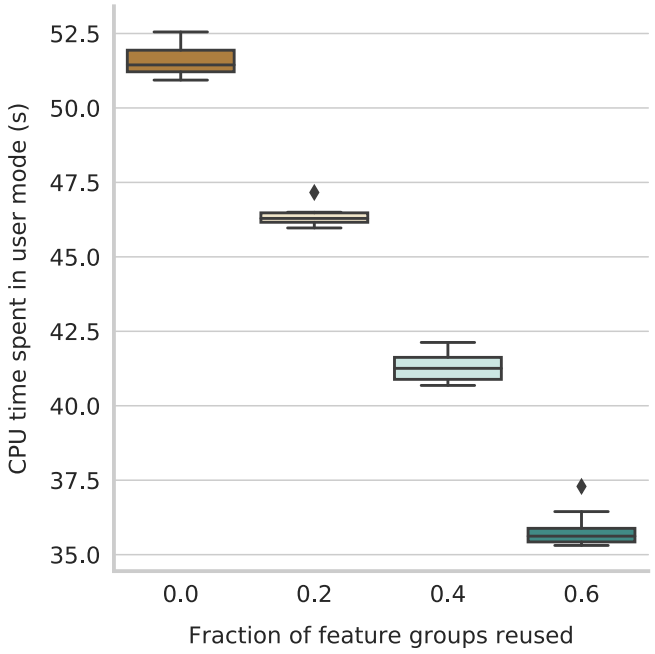
This section will present the results from performing the evaluation episode designed in Section 4.3.4.

5.3.1 Compatibility With Federated Learning

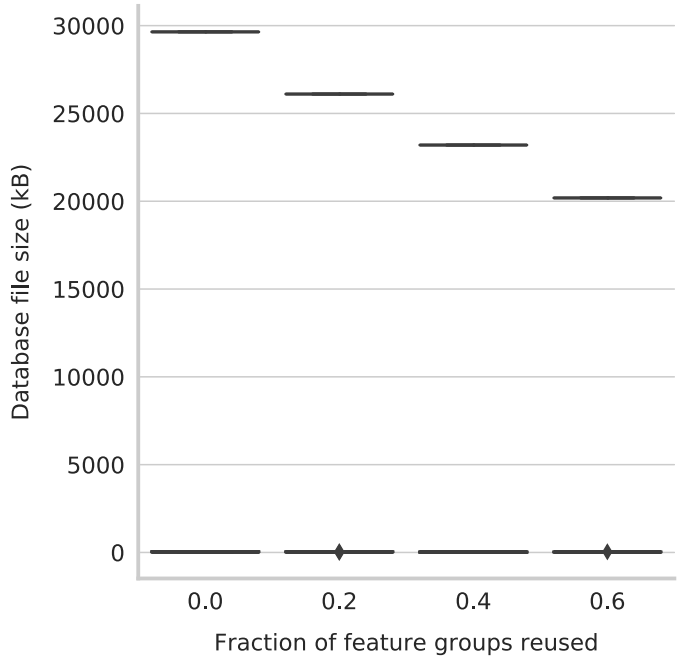
The first part of the evaluation, simulating concurrent FL training with varying levels of feature reuse, demonstrates that the system is compatible with FL and enables feature reuse at different levels between simultaneous FL applications. A logistic regression model is trained with FL on the *Adult* dataset. An SGD regression model is trained with FL on the *Superconductivity* dataset, achieving an R^2 score of 0.7. Models displayed the same performance across all levels of feature reuse, which should be expected as the underlying data used is identical.

5.3.2 Resource Consumption Versus Feature Reuse

The results from the second part of the evaluation, measuring client resource usage during data processing and ingestion of the Adult, Superconductivity, and Wisconsin Breast Cancer datasets, indicate that resource usage decreases when feature reuse increases. A box plot of the CPU time spent in user mode on the simulated client device can be seen in Figure 5.12a. The median CPU time appears to decrease linearly, from 51.4 seconds at 0% reuse to 35.6 seconds at 60% reuse, representing a 30.8% decrease. The database file size shows a similar trend, decreasing apparently linearly from 29644 kB to 20188 kB representing a 31.9% decrease, as seen in the box plot in Figure 5.12b. The database file size has no variance within each level of feature reuse because the data stored is entirely deterministic.



(a) Box plot of CPU time spent in user mode for each level of FG reuse.



(b) Box plot of disk usage for each level of FG reuse as measured by database file size.

Figure 5.12: Cumulative resource usage during Feature Engineering (FE) and feature ingestion at 0%, 20%, 40% and 60% reuse of Feature Groups (FGs).

Table 5.3: Raw data from the resource usage experiment. *Reuse level* is given as a fraction, the *repetition* column indicates the experiment repetition of the given reuse level, UTIME is CPU time spent in user mode, STIME is CPU time spent in system mode. The last column, *database file size kb*, is the size of the database file in kilobytes.

reuse level	repetition	UTIME	STIME	database file size kb
0	1	51.3301	0.645832	29644
0	2	51.5774	0.602151	29644
0	3	52.5518	0.638851	29644
0	4	51.1421	0.665091	29644
0	5	51.2776	0.722154	29644
0	6	52.0592	0.668615	29644
0	7	51.1929	0.641776	29644
0	8	50.935	0.675845	29644
0	9	51.5596	0.675058	29644
0	10	52.2442	0.721064	29644
.2	1	46.2626	0.672219	26104
.2	2	45.9704	0.49879	26104
.2	3	46.4982	0.615765	26104
.2	4	46.1433	0.641844	26104
.2	5	46.457	0.588945	26104
.2	6	46.1394	0.622193	26104
.2	7	46.3187	0.668287	26104
.2	8	46.2129	0.63767	26104
.2	9	47.1615	0.604763	26104
.2	10	46.4841	0.654651	26104
.4	1	41.2704	0.69114	23200
.4	2	40.823	0.530406	23200
.4	3	40.6827	0.565794	23200
.4	4	41.2422	0.515866	23200
.4	5	42.1258	0.612158	23200
.4	6	41.6381	0.608666	23200
.4	7	41.5852	0.547482	23200
.4	8	41.6816	0.508913	23200
.4	9	40.7615	0.584763	23200
.4	10	41.0769	0.59241	23200
.6	1	37.2916	0.557683	20188
.6	2	35.3865	0.528577	20188
.6	3	35.6075	0.469202	20188
.6	4	35.3097	0.558961	20188
.6	5	35.4761	0.501945	20188
.6	6	35.6358	0.538826	20188
.6	7	35.4112	0.47594	20188
.6	8	36.4456	0.498433	20188
.6	9	35.7011	0.378996	20188
.6	10	35.9446	0.426089	20188

5.4 Evaluation of Results

The results presented in Section 5.3 nominally have two implications:

- It is possible to reuse features across different FL applications using the proposed design from Section 5.1.
- Increasing feature reuse reduces resource consumption from FE and feature ingestion on the client.

The resulting system is shown to be compatible with the simulated FL environment on both a regression and a classification problem with structured datasets. Consequently, it can be concluded that the design presented in Section 5.1 is a solution to the RQ of the thesis.

Chapter 6

Discussion

This chapter will evaluate the experimental results from Section 5.3, compare the study results to the related work presented in Chapter 3, discuss implications for industry and academia, and address threats to validity.

6.1 Comparison to Related Work

While the FFS design directly addresses the thesis RQ and some limitations of the related work noted in Section 3.5, it is a highly simplified version of what would likely be expected in a production system, judging by the related work.

6.1.1 Architecture

While virtually all existing FSs are built on a dual-database architecture to enable high throughput for batch queries and low latency for real-time queries, the FFS design in this thesis only specifies a single database. Depending on the specific use case, this could potentially be a performance bottleneck. For cross-device FL (see Section 2.1), where the model is only trained on a few samples per FLC, having both performant batch and singleton queries may not be necessary. Moreover, individual client devices may not even have enough data where database performance would make any significant difference. However, database performance would more likely be an important consideration with cross-silo FL (see Section 2.1), as the data is distributed among a small number of clients. This is further discussed in Section 6.2.

6.1.2 Software Features

Existing FSs also typically have some support for versioning FE code together with the corresponding feature definition and executing it on ingested data, ensuring that the same code is always used to generate features. This has not been considered for the FFS design and could present a limitation compared with related work.

Although the artifact enables feature reuse from a technical perspective, it lacks the possibility for users to browse and discover existing feature definitions. A browsable registry letting users discover feature definitions and understand their data sources and business rules is crucial for facilitating feature reuse in an organization [31].

Organizations with multiple teams may want access-controlled feature definitions, as client data in FL is typically privacy-sensitive. This is offered in some existing FSs, such as Hopsworks, but is not addressed in the FFS design. Access-controlled feature definitions could be desirable in organizations with restrictions for which teams can use certain features for their models.

Many existing FSs have time-travel functionality, i.e., being able to access the feature values of different points in the past, essentially treating every feature as a time-series. This solves the problem of future information leaking into training data, but is something not considered in the FFS design, representing a limitation compared to related work.

6.2 Applications to Different Types of Federated Learning

The design of the FFS is first and foremost aimed at HFL, as it assumes that all participants in an FL task will use the same set of features. However, this is not the case for VFL applications. Participants in a VFL application have different features, meaning that defining datasets and FGs in the FR as done in the FFS may not make sense in practice. However, it may still be desirable to centrally manage definitions analogously to how a single CS manages the FL training. Further study is needed to determine a good way of achieving this for VFL, but it will likely require modifications in both the data model and architecture. Additionally, encrypted entity alignment (see Section 2.1) could be integrated as part of the FFS. Whether alignment should fall under the responsibilities of the FFS is not clear. However, as this is required for all VFL applications, it seems natural that it should at least be abstracted away from the specific FL application. If the FFS design can be adapted to accommodate both HFL and VFL, it could also open the door to use with federated transfer learning.

For large volumes of feature data, which may be expected in cross-silo applications, the single-database design may result in unsatisfactory performance, particularly for multitenant applications. The cross-silo FL scenario is similar to the conventional centralized ML, in which the FSs need both low-latency singleton queries for operations and high-throughput batch queries for training. The reliance on a single database, rather than the online/offline database design usually found in FSs, means that the FFS is currently better suited for cross-device FL scenarios, where the data volume may be significantly lower. More importantly, communications represent a much more significant performance bottleneck for cross-device FL as a result of high-dimensionality model updates and limited communications bandwidth for participating devices. The inherent difference in resource constraints between cross-silo and cross-device FL likely means that the

Table 6.1: Summary of suitability of the Federated Feature Store (FFS) to different types of Federated Learning (FL). 1: suitable, 0: may need accommodation, N/A: not applicable. Cross-silo and cross-device mainly describe participant characteristics, and are therefore mostly affected by system architecture. The FFS design is most suitable to cross-device Horizontal FL (HFL) applications.

Type	Data model	Architecture	Software features
HFL	1	1	1
VFL	0	0	0
Cross-silo	N/A	0	N/A
Cross-device	N/A	1	N/A

two types require different FS architectures.

Table 6.1 summarizes the suitability of the FFS to different types of FL in terms of the data model, the architecture, and the software features.

6.3 Implications to Industry

While the FFS is far from ready for production, it demonstrates the design for a novel application of the already established and industry-proven FS concept. Given the assumptions presented in Chapter 2, namely that we will see growing adoption of FL, the presented design can serve as a starting point for implementing a production-ready system for reusing ML features in an FL context. This would be useful to organizations with multiple FL applications per client device, particularly for larger organizations that use thousands of different features, such as those mentioned in Section 2.3. The method of adoption would likely depend on the hardware/software environment of the client devices in use. While devices with more limited capabilities may need more specialized designs/implementations, scenarios with more performant hardware, typically cross-silo FL, could use existing FSs adapted for FL.

The evaluation results showing a decrease in resource usage with increased feature reuse indicate some benefits businesses may reap from eliminating duplicate features. The reduced resource consumption, and hence lower energy consumption, may facilitate businesses in complying with legislation and shareholder demands for ESG and give more legitimacy to their commitment to going green and reducing environmental impact. Further, the lower energy and resource consumption also results in lower operating expenditures for businesses through reduced electricity or cloud computing bills.

6.4 Implications to Academia

The evaluation demonstrated that a higher degree of feature reuse results in lower resource consumption in terms of both CPU and disk usage. While this is a goal

unto itself for resource-constrained devices, other benefits of more feature reuse should also be studied. Research into the environmental and economic benefits of feature reuse as a consequence of lower energy consumption should be investigated, in addition to studying the economic benefits of eliminating duplicate FE efforts. Future studies should also examine how feature reuse scales with organizations in terms of the number of models, features, and people. While larger organizations such as Uber and Facebook report substantial benefits in being able to reuse features (see Section 2.3), it is not known how well it scales down to smaller organizations.

Though the FFS design is reasonably suited to HFL, it may not be practical for the purposes of VFL, as discussed in Section 6.2. Further studies should be conducted to better enable feature reuse for VFL. Additionally, Further, secure entity alignment should be investiatged as apossible responsibility of the FFS, or more generally as a separate

The more general case of edge ML, of which FL can be viewed as a subset, could potentially use a similar design as the one proposed in this thesis. With more relaxed constraints on data privacy, there could be opportunities to bring FSs to edge devices and servers, thereby reducing latency and improving quality of service.

As FL typically involves sensitive data, privacy and security should not be taken lightly. Researchers should therefore study the implications of feature reuse and FSs to security and privacy of participant data.

6.5 Threats to Validity

6.5.1 Threats to Internal Validity

As the artifact was only tested in a simulated FL environment on a local machine, compatibility with FL in a real-world scenario may not be conclusive. The FL framework used is built for real-world use cases. It has been tested on millions of clients on a single machine [61] and with physical mobile and edge devices [60] (albeit with a maximum of 10 devices), so any uncertainty of FL compatibility would mainly lie with the internal communications of the FFS, between CFSs and the FR. The method of internal communications is left as an implementation detail in the design, as this should generally make no functional difference in the system's operation. The implemented artifact, like the FL framework, uses RPC for communications, so it should be reasonable to conclude that the design is at least structurally compatible with FL.

CPU usage was measured with the `getrusage` system call, which has a low level of overhead and intrusiveness, i.e., it imposes little performance degradation and task interference [62]. While the specific `getrusage` implementation only returned time with millisecond precision, this should not be an issue, as the measured tasks ran on the order of tens of seconds (see Section 5.3).

Network communications for downloading dataset and FG definitions were

included in the measured CPU time, but this should not skew the measurements to a significant degree for two reasons: 1) the number of network requests is constant for all levels of feature reuse, as the same number of datasets were used each time, and 2) the connection was purely local, i.e., no packets ever left the machine, meaning that network latency is a negligible factor.

The evaluated artifact is a naïve implementation of the design and is only tested in a simulated environment and does not adhere to any constraints on security, privacy, etc. It is not a given that implementing the design while adhering to more realistic requirements would yield a system exhibiting the same resource usage trends or compatibility with FL.

6.5.2 Threats to External Validity

Perhaps the biggest threat to the external validity of the thesis is that it rests on the assumption that FL use will increase in the future, so the presented artifact and design are solutions to an as of yet hypothetical (or at least unreported) problem. Consequently, the results are not immediately useful or relevant to any real-world applications. However, given the growth observed in edge devices, ML, and data privacy concerns, as discussed in Section 2.1, the future of FL seems promising.

The evaluation strictly used structured datasets (the only type of data the artifact was implemented to handle). In contrast, most papers on FL use other types of data, such as images and text [48]. However, the literature consists of few applications of FL in production, as most papers describe applications at the proof-of-concept stage, so this may not be a big concern for real-world usage. Moreover, there is no apparent limitation on possible feature data types in the presented architecture, so it should be possible to implement a system supporting other types of data based on the FFS design.

Chapter 7

Conclusions and Further Work

This thesis presents a novel design for a system enabling reuse of ML features in FL, a previously unserved area of ML by existing FS solutions. A proof of concept is implemented to establish the feasibility of the design. Experimental results show decreasing levels of resource usage for the client device with increasing levels of feature reuse, indicating a performance benefit when reducing duplicate feature management efforts through reusing existing features. The decreased resource usage also has implications for green IT and ESG, allowing organizations to decrease their negative environmental impact by reducing energy consumption through feature reuse.

While the results suggest that applying the FS concept to the problem of FL is feasible, there are still many possible directions for further work to increase the utility of the system: The FR should be browsable to make feature discovery easy for data scientists and other practitioners, the system should have support for versioning and executing FE code, it should be possible to create derived features, automatic cleanup of old data to limit disk usage on resource-constrained devices, time-travel capabilities, etc.

Bibliography

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017. JMLR: W&CP volume 54*, Feb. 2016. arXiv: 1602.05629 [cs.LG].

[2] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, Jul. 2015. DOI: 10.1126/science.aaa8415.

[3] I. Dayan, H. R. Roth, A. Zhong, A. Harouni, A. Gentili, A. Z. Abidin, A. Liu, A. B. Costa, B. J. Wood, C.-S. Tsai, C.-H. Wang, C.-N. Hsu, C. K. Lee, P. Ruan, D. Xu, D. Wu, E. Huang, F. C. Kitamura, G. Lacey, G. C. de Antônio Corradi, G. Nino, H.-H. Shin, H. Obinata, H. Ren, J. C. Crane, J. Tetreault, J. Guan, J. W. Garrett, J. D. Kaggie, J. G. Park, K. Dreyer, K. Juluru, K. Kersten, M. A. B. C. Rockenbach, M. G. Linguraru, M. A. Haider, M. AbdelMaseeh, N. Rieke, P. F. Damasceno, P. M. C. e Silva, P. Wang, S. Xu, S. Kawano, S. Sriswasdi, S. Y. Park, T. M. Grist, V. Buch, W. Jantarabenjakul, W. Wang, W. Y. Tak, X. Li, X. Lin, Y. J. Kwon, A. Quraini, A. Feng, A. N. Priest, B. Turkbey, B. Glicksberg, B. Bizzo, B. S. Kim, C. Tor-Díez, C.-C. Lee, C.-J. Hsu, C. Lin, C.-L. Lai, C. P. Hess, C. Compas, D. Bhatia, E. K. Oermann, E. Leibovitz, H. Sasaki, H. Mori, I. Yang, J. H. Sohn, K. N. K. Murthy, L.-C. Fu, M. R. F. de Mendonça, M. Fralick, M. K. Kang, M. Adil, N. Gangai, P. Vateekul, P. Elnajjar, S. Hickman, S. Majumdar, S. L. McLeod, S. Reed, S. Gräf, S. Harmon, T. Kodama, T. Puthanakit, T. Mazzulli, V. L. de Lavor, Y. Rakvongthai, Y. R. Lee, Y. Wen, F. J. Gilbert, M. G. Flores, and Q. Li, "Federated learning for predicting clinical outcomes in patients with COVID-19," *Nature Medicine*, vol. 27, no. 10, pp. 1735–1743, Sep. 2021. DOI: 10.1038/s41591-021-01506-3.

[4] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for internet of things: Recent advances, taxonomy, and open challenges," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1759–1799, 2021. DOI: 10.1109/comst.2021.3090430.

[5] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106 854, Nov. 2020. DOI: 10.1016/j.cie.2020.106854.

[6] G. Giray, "A software engineering perspective on engineering machine learning systems: State of the art and challenges," *Journal of Systems and Software*, vol. 180, p. 111 031, Oct. 2021. DOI: 10.1016/j.jss.2021.111031.

[7] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," *ACM Computing Surveys*, arXiv:2011.09926, Apr. 2022. DOI: 10.1145/3533378. arXiv: 2011.09926 [cs.LG]. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2020arXiv201109926P.

[8] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, 2015.

[9] L. E. Li, E. Chen, J. Hermann, P. Zhang, and L. Wang, "Scaling machine learning as a service," in *Proceedings of The 3rd International Conference on Predictive Applications and APIs*, C. Hardgrove, L. Dorard, K. Thompson, and F. Douetteau, Eds., ser. Proceedings of Machine Learning Research, vol. 67, PMLR, Oct. 2017, pp. 14–29. [Online]. Available: https://proceedings.mlr.press/v67/li17a.html.

[10] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, Feb. 2018. DOI: 10.1109/hpca.2018.00059.

[11] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, May 2019. DOI: 10.1109/icse-seip.2019.00042.

[12] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," Nov. 2018. arXiv: 1811.03604 [cs.CL].

[13] M. Ismail, E. Gebremeskel, T. Kakantousis, G. Berthou, and J. Dowling, "Hopsworks: Improving user experience and development on hadoop with scalable, strongly consistent metadata," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, Jun. 2017. DOI: 10.1109/icdcs.2017.41.

[14] Feast, *Feast 0.21 documentation*, Accessed May 30, 2022, 2022. [Online]. Available: https://docs.feast.dev/.

[15] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[16] K. Hamidieh, "A data-driven statistical model for predicting the critical temperature of a superconductor," *Computational Materials Science*, vol. 154, pp. 346–354, Nov. 2018. DOI: 10.1016/j.commatsci.2018.07.052.

[17] K. P. Bennett and O. L. Mangasarian, "Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods and Software*, vol. 1, no. 1, pp. 23–34, Jan. 1992. DOI: 10.1080/10556789208805504.

[18] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch, and H. H. Olsson, "Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions," *Information and Software Technology*, vol. 127, p. 106 368, Nov. 2020. DOI: 10.1016/j.infsof.2020.106368.

[19] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1, Dec. 2021. DOI: 10.1561/2200000083. arXiv: 1912.04977 [cs.LG].

[20] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, Mar. 2019. DOI: 10.1145/3298981.

[21] K. Wei, J. Li, C. Ma, M. Ding, S. Wei, F. Wu, G. Chen, and T. Ranbaduge, "Vertical federated learning: Challenges, methodologies and experiments," Feb. 2022. arXiv: 2202.04309 [cs.LG].

[22] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020. DOI: 10.1109/comst.2020.2986024.

[23] A. Huang, Y. Liu, T. Chen, Y. Zhou, Q. Sun, H. Chai, and Q. Yang, "StarFL: Hybrid federated learning architecture for smart urban computing," *ACM Transactions on Intelligent Systems and Technology*, vol. 12, no. 4, pp. 1–23, Aug. 2021. DOI: 10.1145/3467956.

[24] F. Hartmann, S. Suh, A. Komarzewski, T. D. Smith, and I. Segall, "Federated learning for ranking browser history suggestions," Nov. 2019. arXiv: 1911.11807 [cs.LG].

[25] B. Tan, B. Liu, V. Zheng, and Q. Yang, "A federated recommender system for online services," in *Fourteenth ACM Conference on Recommender Systems*, ACM, Sep. 2020. DOI: 10.1145/3383313.3411528.

[26] F. Kumeno, "Sofware engneering challenges for machine learning applications: A literature review," *Intelligent Decision Technologies*, vol. 13, pp. 463–476, 2020, ISSN: 18724981, 18758843. DOI: 10.3233/IDT-190160.

[27] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, and S. Wagner, "Software engineering for AI-based systems: A survey," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 2, pp. 1–59, Apr. 2022. DOI: 10.1145/3487043. arXiv: 2105.01984 [cs.SE].

[28] L. Orr, A. Sanyal, X. Ling, K. Goel, and M. Leszczynski, "Managing ML pipelines," *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 3178–3181, Jul. 2021. DOI: 10.14778/3476311.3476402. arXiv: 2108.05053 [cs.LG].

[29] T. Kakantousis, A. Kouzoupis, F. Buso, G. Berthou, J. Dowling, and S. Haridi, "Horizontally scalable ml pipelines with a feature store," in *Proc. 2nd SysML Conf., Palo Alto, USA*, 2019. [Online]. Available: https://mlsys.org/Conferences/2019/doc/2019/demo_7.pdf.

[30] J. Hermann and M. Del Balso, *Meet michelangelo: Uber's machine learning platform*, Accessed May 30, 2022, Sep. 2017. [Online]. Available: https://eng.uber.com/michelangelo.

[31] J. Patel, "The democratization of machine learning features," in *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, IEEE, Aug. 2020. DOI: 10.1109/iri49571.2020.00027.

[32] V. Mour, S. Dey, S. Jain, and R. Lodhe, "Feature store for enhanced explainability in support ticket classification," in *Natural Language Processing and Chinese Computing*, Springer International Publishing, 2020, pp. 467–478. DOI: 10.1007/978-3-030-60457-8_38.

[33] A. Khan and Z. S. Hassan, *Building a gigascale ml feature store with redis, binary serialization, string hashing, and compression*, DoorDash Engineering, Nov. 2020. [Online]. Available: https://doordash.engineering/2020/11/19/building-a-gigascale-ml-feature-store-with-redis/.

[34] G. Cerar, B. Bertalanič, A. Pirnat, A. Čampa, and C. Fortuna, "On designing data models for energy feature stores," May 2022. arXiv: 2205.04267 [cs.AI].

[35] S. Murugesan, "Harnessing green IT: Principles and practices," *IT Professional*, vol. 10, no. 1, pp. 24–33, 2008, ISSN: 1520-9202. DOI: 10.1109/mitp.2008.10.

[36] G. Luo, B. Guo, Y. Shen, H. Liao, and L. Ren, "Analysis and optimization of embedded software energy consumption on the source code and algorithm level," in *2009 Fourth International Conference on Embedded and Multimedia Computing*, IEEE, Dec. 2009. DOI: 10.1109/em-com.2009.5402965.

[37] G. Friede, T. Busch, and A. Bassen, "Esg and financial performance: Aggregated evidence from more than 2000 empirical studies," *Journal of Sustainable Finance & Investment*, vol. 5, no. 4, pp. 210–233, Oct. 2015, ISSN: 2043-0795. DOI: 10.1080/20430795.2015.1118917.

[38] R. Lozano, "A holistic perspective on corporate sustainability drivers," *Corporate Social Responsibility and Environmental Management*, vol. 22, no. 1, pp. 32–44, Apr. 2013. DOI: 10.1002/csr.1325.

[39] D. H. Hagos, T. Kakantousis, V. Vlassov, S. Sheikholeslami, T. Wang, J. Dowling, C. Paris, D. Marinelli, G. Weikmann, L. Bruzzone, S. Khaleghian, T. Kraemer, T. Eltoft, A. Marinoni, D.-A. Pantazi, G. Stamoulis, D. Bilidas, G. Papadakis, G. Mandilaras, M. Koubarakis, A. Troumpoukis, S. Konstantopoulos, M. Muerth, F. Appel, A. Fleming, and A. Cziferszky, "ExtremeEarth meets satellite data from space," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 9038–9063, 2021. DOI: 10.1109/jstars.2021.3107982.

[40] A. A. Ormenisan, M. Meister, F. Buso, R. Andersson, S. Haridi, and J. Dowling, "Time travel and provenance for machine learning pipelines," in *2020 USENIX Conference on Operational Machine Learning (OpML 20)*, USENIX Association, Jul. 2020. [Online]. Available: https://www.usenix.org/conference/opml20/presentation/ormenisan.

[41] Hopsworks, *Hopsworks 2.5.9 feature store documentation*, Accessed May 30, 2022, 2022. [Online]. Available: https://docs.hopsworks.ai/feature-store-api/2.5.9/.

[42] P. Fang, Z. Cai, H. Chen, and Q. Shi, "Flfe: A communication-efficient and privacy-preserving federated feature engineering framework," Sep. 2020. arXiv: 2009.02557 [cs.LG].

[43] L. Wang, G. Luo, K. Yi, and G. Cormode, "Quantiles over data streams," in *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, ACM Press, 2013. DOI: 10.1145/2463676.2465312.

[44] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, Dec. 2007. DOI: 10.2753/mis0742-1222240302.

[45] Hevner, March, Park, and Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, p. 75, 2004. DOI: 10.2307/25148625.

[46] J. Venable, J. Pries-Heje, and R. Baskerville, "FEDS: A framework for evaluation in design science research," *European Journal of Information Systems*, vol. 25, no. 1, pp. 77–89, Jan. 2016. DOI: 10.1057/ejis.2014.36.

[47] J. F. Nunamaker, M. Chen, and T. D. M. Purdin, "Systems development in information systems research," *Journal of Management Information Systems*, vol. 7, no. 3, pp. 89–106, Dec. 1990, ISSN: 0742-1222. DOI: 10.1080/07421222.1990.11517898.

[48] S. K. Lo, Q. Lu, C. Wang, H.-Y. Paik, and L. Zhu, "A systematic literature review on federated machine learning, From a software engineering perspective," *ACM Computing Surveys*, vol. 54, no. 5, pp. 1–39, Jun. 2022, ISSN: 0360-0300. DOI: 10.1145/3450288.

[49] C. Philippenko and A. Dieuleveut, "Preserved central model for faster bidirectional compression in distributed settings," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[50] S. Hong and J. Chae, "Distributed online learning with multiple kernels," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021. DOI: 10.1109/tnnls.2021.3105146.

[51] A. Das and S. Patterson, "Multi-tier federated learning for vertically partitioned data," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, Jun. 2021. DOI: 10.1109/icassp39728.2021.9415026.

[52] Y. Liu, Y. Liu, Z. Liu, Y. Liang, C. Meng, J. Zhang, and Y. Zheng, "Federated forest," *IEEE Transactions on Big Data*, vol. 8, no. 3, pp. 843–854, Jun. 2022. DOI: 10.1109/tbdata.2020.2992755.

[53] J. Park, D. Kwon, and S. hong, "Ofedqit: Communication-efficient online federated learning via quantization and intermittent transmission," May 2022. arXiv: 2205.06491 [cs.LG].

[54] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Jun. 2019, pp. 4615–4625. [Online]. Available: https://proceedings.mlr.press/v97/mohri19a.html.

[55] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Jun. 2019, pp. 634–643. [Online]. Available: https://proceedings.mlr.press/v97/bhagoji19a.html.

[56] L. Lyu, X. Xu, Q. Wang, and H. Yu, "Collaborative fairness in federated learning," in *Lecture Notes in Computer Science*, Springer International Publishing, 2020, pp. 189–204. DOI: 10.1007/978-3-030-63076-8_14.

[57] S. Cui, W. Pan, J. Liang, C. Zhang, and F. Wang, "Addressing algorithmic disparity and performance inconsistency in federated learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 26 091–26 102. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/db8e1af0cb3aca1ae2d0018624204529-Paper.pdf.

[58] Z. Hu, K. Shaloudegi, G. Zhang, and Y. Yu, "Federated learning meets multi-objective optimization," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2022. DOI: 10.1109/tnse.2022.3169117.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[60] A. Mathur, D. J. Beutel, P. P. B. de Gusmão, J. Fernandez-Marques, T. Topal, X. Qiu, T. Parcollet, Y. Gao, and N. D. Lane, "On-device federated learning with flower," *On-device Intelligence Workshop at the Fourth Conference on Machine Learning and Systems (MLSys)*, Apr. 2021. DOI: 10.48550/ARXIV.2104.03042.

[61] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," Jul. 2020. arXiv: 2007.14390 [cs.LG].

[62] G. Juve, B. Tovar, R. F. D. Silva, D. Krol, D. Thain, E. Deelman, W. Allcock, and M. Livny, "Practical resource monitoring for robust high throughput computing," in *2015 IEEE International Conference on Cluster Computing*, IEEE, Sep. 2015. DOI: 10.1109/cluster.2015.115.

Appendix A

API Specification

A.1 Registry API

A.1.1 `Registry.create_feature_group`

Arguments

- `name` – feature group name
- `version` – feature group version number
- `description` – description of feature group
- `dataframe` – dataframe with a sample data

The `create_feature_group` method creates a new feature group definition with the given metadata. A `CREATE TABLE` statement is generated automatically based on data types inferred from the `dataframe` argument.

A.1.2 `Registry.create_dataset`

Arguments:

- `name` – dataset name
- `version` – dataset version number
- `description` – dataset description
- `training_sql` – SQL query for retrieving training data
- `inference_sql` – SQL query for retrieving feature data during inference
- `feature_group_dependencies` – names of feature groups used by this dataset
- `target_variable` – name of target variable for inference

Creates a new dataset definition.

A.2 Client Feature Store API

A.2.1 `ClientFeatureStore.get_dataset_instance`

Arguments:

- `name` – dataset name
- `version` – dataset version number

Returns a `ClientDataset` instance which can be used to efficiently retrieve batches of training samples.

A.2.2 `ClientFeatureStore.insert_feature_data`

- `feature_group_name` – name of feature group to insert data for
- `feature_group_version` – version of feature group
- `df` – dataframe containing the feature data to be inserted

Inserts data from `df` into the feature group `feature_group_name` in the local database.

A.2.3 `ClientDataset.get_training_data`

Arguments:

- `batch_size` – number of samples to return per batch

Returns a generator yielding batches of size `batch_size` with training samples.

Appendix B

Database Schemas

B.1 Client Feature Store Database Schema

Name	Datatype	Description
name	string	Name of dataset
description	string	Dataset description
sql_get_training_samples_query	string	SQL query for fetching training samples during training
id	integer	Internal primary key for convenience

Table B.1: Schema for the dataset table on the Client Feature Store (CFS), which is identical to the dataset table on the Feature Registry (FR)

B.2 Feature Registry Database Schema

Name	Datatype	Description
name	string	Name of feature group
description	string	Description of the feature group
sql_create_statement	string	SQL statement for creating the feature group table on the clients
id	integer	Internal primary key for convenience

Table B.2: Database schema for the feature group definition table on the FR.

Name	Datatype	Description
name	string	Name of dataset
description	string	Dataset description
sql_get_training_samples_query	string	SQL query for fetching training samples during training
id	integer	Internal primary key for convenience

Table B.3: Schema for the dataset table on the Feature Registry (FR).

Name	Datatype	Description
feature_group_id	integer	ID of the feature group
dataset_id	integer	ID of the dataset

Table B.4: Schema for the table representing the many-to-many relationship between feature groups and datasets in the Feature Registry (FR). feature_group_id is a foreign key to the feature group definition table Table B.2 and dataset_id is a foreign key to the dataset table Table B.3.

NTNU

Norwegian University of
Science and Technology