ExDRa: Exploratory Data Science on Federated Raw Data

Sebastian Baunsgaard[3], Matthias Boehm[3], Ankit Chaudhary[4], Behrouz Derakhshan[2]
Stefan Geißelsöder[1], Philipp M. Grulich[4], Michael Hildebrand[1], Kevin Innerebner[3]
Volker Markl[2,4], Claus Neubauer[1], Sarah Osterburg[1], Olga Ovcharenko[3], Sergey Redyuk[4]
Tobias Rieger[3], Alireza Rezaei Mahdiraji[2], Sebastian Benjamin Wrede[3], Steffen Zeuch[2]

[1] Siemens AG; Berlin/Erlangen, Germany
[2] DFKI GmbH; Berlin, Germany
[3] Graz University of Technology; Graz, Austria
[4] Technische Universität Berlin; Berlin, Germany

ABSTRACT

Data science workflows are largely exploratory, dealing with underspecified objectives, open-ended problems, and unknown business value. Therefore, little investment is made in systematic acquisition, integration, and pre-processing of data. This lack of infrastructure results in redundant manual effort and computation. Furthermore, central data consolidation is not always technically or economically desirable or even feasible (e.g., due to privacy, and/or data ownership). The ExDRa system aims to provide system infrastructure for this exploratory data science process on federated and heterogeneous, raw data sources. Technical focus areas include (1) ad-hoc and federated data integration on raw data, (2) data organization and reuse of intermediates, and (3) optimization of the data science lifecycle, under awareness of partially accessible data. In this paper, we describe use cases, the overall system architecture, selected features of SystemDS' new federated backend (for federated linear algebra programs, federated parameter servers, and federated data preparation), as well as promising initial results. Beyond existing work on federated learning, ExDRa focuses on enterprise federated ML and related data pre-processing challenges. In this context, federated ML has the potential to create a more fine-grained spectrum of data ownership and thus, even new markets.

ACM Reference Format:
Sebastian Baunsgaard[3], Matthias Boehm[3], Ankit Chaudhary[4], Behrouz Derakhshan[2], Stefan Geißelsöder[1], Philipp M. Grulich[4], Michael Hildebrand[1], Kevin Innerebner[3], Volker Markl[2,4], Claus Neubauer[1], Sarah Osterburg[1], Olga Ovcharenko[3], Sergey Redyuk[4], and Tobias Rieger[3], Alireza Rezaei Mahdiraji[2], Sebastian Benjamin Wrede[3], Steffen Zeuch[2]. 2021. ExDRa: Exploratory Data Science on Federated Raw Data. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3448016.3457549

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457549

1 INTRODUCTION

Data Science Workflows: Typical data science projects deal with open-ended questions, aiming to find patterns and predictive models that create business value [8]. Data scientists acquire samples for the given problem, and then enrich, clean, and prepare these data for model building and hypothesis testing [16]. This process is largely exploratory, where analysis results guide the incremental refinement of ML pipelines [23, 94]. Due to the unknown value and exploratory character—in stark contrast to traditional data warehousing [16]—little investment is made in systematic data acquisition, integration, and pre-processing [76].

Query Processing on Raw Data: The inspiring work on query processing on raw data [3, 35, 47] supports exploratory data analysis by executing SQL queries directly on raw data files. Repeated data access overhead is alleviated with dedicated caching strategies, positional maps for partial parsing, partitioning, and code generation. However, executing entire ML pipelines on raw data—with data cleaning and model training—remains an open problem.

Data Ownership and Privacy: Besides the exploratory character, a second major challenge is the limited access to data sources relevant to the given business problems. First, privacy-preserving data storage may not allow for central data consolidation. Similarly, in geo-distributed applications, central data consolidation can be economically or technically infeasible [89]. Second, data ownership limitations restrict the scope of data enrichment and consolidation.

Example 1 (Data Ownership Dilemma). *Consider a scenario of a machine vendor, a middle person who uses provided machines to test customer equipment, and customers. Any one of these stakeholders has vested interest in owning the detailed test measurements. The machine vendor wants to model and improve operations; the customer does not want to reveal equipment characteristics; and the middle person wants to offer data-driven products to various customers.*

Therefore, data ownership and sharing agreements are usually negotiated in bilateral contracts among stakeholders. In the context of exploratory data science, these access limitations are, however, challenging because the return on investment (e.g., impact on predictions) of data purchases or subscriptions is unknown upfront.

Federated ML: Recent work on federated learning [9] partially addresses this problem of inaccessible data. Federated ML learns a global model without central data consolidation, for example, on private data of mobile devices. Existing work adopts a traditional parameter server architecture [20, 41, 53, 79]. The parameter server

initializes and broadcasts a model, every client device now acts as a worker, computes gradients over mini-batches of their local data (e.g., via a forward and backward pass through a neural network), and pushes these gradients back to the server, where they are aggregated into a model update. This process is repeated until convergence, with client sampling and distributed aggregation for robustness. However, this infrastructure is tailor-made—and primarily applicable—for pre-processing via static embeddings, mini-batch DNN algorithms, and mobile client devices. We see a much broader opportunity for federated learning.

Enterprise Federated ML: The ExDRa system [1] addresses these issues by introducing a system infrastructure for exploratory data science on heterogeneous, structured, and only partially accessible raw data. To this end, we combine federated ML for batch and mini-batch training with query processing on raw data. Major challenges include data cleaning and preparation on raw data, support for both linear algebra programs and parameter servers, as well as automatic plan optimization and federated data re-organization for eliminating unnecessary redundancy. Overall, we see many opportunities for *enterprise federated ML*, where ML pipelines are executed on federated, structured input data by exchanging aggregates that do not reveal the underlying private federated data. This approach creates a new spectrum of data ownership and data/model sharing (without the danger of data redistribution or reselling), and thus, has the potential to create new markets.

Contributions: In this paper, we describe the overall ExDRa system architecture for federated ML in the enterprise, and share insights and results from the first years of building this infrastructure. Our detailed contributions are:

- *Use Cases:* We introduce exploratory and deployment use cases in Section 2. Beside these example ML applications, we also characterize types of federated data, privacy models, and deployments for federated ML on raw data.
- *System Architecture:* In Section 3, we describe the overall system architecture and its components for model and pipeline management, federated ML and data preparation, as well as streaming data acquisition. Section 5 then also describes the envisioned deployment in production.
- *Federated ML Runtime:* Section 4 describes in detail the federated runtime backend of Apache SystemDS, federated data organization, as well as federated linear algebra, federated parameter servers, and federated data transformations.
- *Preliminary Results:* Our experiments in Section 6 show promising results of federated ML algorithms and pipelines, with low to moderate overhead compared to local training.

2 USE CASES

As a motivation for enterprise federated learning, we introduce two production use cases from the process industry, and then characterize federated data, privacy models, and deployment types.

2.1 Fertilizer Production

Context: The Food and Agriculture Organization of the United Nations (FAO) estimated the worldwide phosphorus-based fertilizer nutrient consumption of 2020 as 45 million tons [27]. The manufacturing process of those fertilizers entails grinding phosphorus rock. To increase fertilizer output, companies want to detect erroneous behavior as soon as possible. However, good maintenance processes cause class imbalance in terms of rare negative samples (e.g., failures or anomalies), which makes it challenging to train robust predictive models. Leveraging the data from multiple, federated sites could thus, enhance the data quality significantly.

Data: There are three crucial inputs to the mill at the grinding facility: (1) the cleaned raw material (metal removed), (2) roll force to crack the raw material into target granularity, and (3) hot gases evaporating the moisture trapped in the material. To ensure high input quality, the settings and behavior of each subunit—like storage, rollers, ventilation, exhaust gas and roll lubrication systems—are constantly monitored. In detail, we record physical and electrical parameters from 68 sensors at 1-second granularity. Sensor measurements include power, currents, temperatures, pressure differences, tank levels, conveyance speeds, vibration frequencies, air flows, humidity and weights of different parts of the production line. Finally, these grinding mill data are used to create unsupervised anomaly detection models (e.g., Gaussian mixture models).

2.2 Paper Production

Context: In paper production, quality is measured by quantities such as z-strength and Scott Bond [49]. If these values are outside the specification, the paper is assigned a lower quality class. Alternatively, the paper is shredded and reintroduced into the production process. Both of these negative outcomes reduce the potential revenue, where the second option further increases the consumption of water, chemicals and electricity. The quality can only be measured at the end of the process, after winding up the paper. This measurement is performed approximately every 45 minutes and takes up to 20 minutes. As a result, there is a substantial delay between the production start and quality measurement. Therefore, predicting the z-strength for certain recipes and configurations during production can shorten the reaction time in case of quality problems, and thus, reduce the amount of paper with lower quality.

Data: The production process consists of four steps: initial pulp mixing, grinding, chemical treatment, and rolling. Each step impacts paper quality with a varying time delay from seconds (rolling) up to one day (pulp mixing). Throughout the process, 97 signals are recorded with different sampling rates. The data is extracted from a dedicated distributed control system (DCS) and process databases. In detail, measured variables include indicators for pulp quality, granularity, powers, inflows and compositions of various chemicals, recipe IDs, humidity, and machine characteristics such as speeds and torques of various rolls. These data signals are preprocessed and used for training neural network regression models that predict paper quality and classify specification violations.

2.3 Enterprise Federated ML

There are many enterprise federated ML use cases similar to our two examples from Sections 2.1 and 2.2. In this section, we aim to generally characterize such use cases in terms of federated data types, privacy models, pipeline characteristics, and collaboration schemes. Similar to permissioned blockchains [6, 33, 62], in such

[1] ExDRa Project: https://www.exdra.de/en/

enterprise federated ML use cases, we can presume legal agreements and coordination among involved parties.

Federated Data: In a federated environment, the raw, detailed data remains at the individual federated sites. Operations are pushed down to these sites to allow for global data analysis and model building. Conceptually, a virtual, federated frame, matrix or tensor is composed of arbitrary non-overlapping, i.e., disjoint, regions pointing to data at the federated sites. Two important special cases are predominant in practice though:

- *Row-Partitioned* federated data—or horizontal federated learning [9, 95]—refers to partitions of rows, where every federated site holds a subset of observations. In time series scenarios, an observation might be a subsequence of certain length. These row partitions share the same original features, but might not contain all categories of a categorical feature (and thus, one-hot encoded columns) or no values for a site-specific feature. The labels for supervised learning might exist at one or multiple sources, or at the coordinator.
- *Column-Partitioned* federated data—or vertical federated learning [92, 95]—is a less common form that refers to partitions of columns, where every federated site holds a—potentially overlapping—subset of features. Examples are site-specific measurement processes (e.g., available sensors), and detailed material information along the supply chain, which can be spatial-temporally joined with the measurements to construct an enriched federated feature matrix.

Federated Privacy Models: Federated learning fundamentally aims to allow training and scoring of ML models without central data consolidation. A key desirable property is that shared information does not allow reconstructing the private raw data of federated sites. In contrast to fully untrusted environments, for enterprise federated ML, there is spectrum of means for preventing reconstruction with different privacy guarantees and performance characteristics:

- *Aggregates:* Many ML models can be learned in a federated environment by restricting the communication to aggregates (e.g., gradients). If these aggregates include sufficiently many observations and/or features, such aggregates share information on distributions but do not reveal the raw data.
- *Encrypted Communication:* In addition to aggregation, the communication channels might need—unless otherwise covered—protection through traditional encryption methods to ensure the aggregates are only shared with trusted parties.
- *Privacy Enhancing Technologies:* If even aggregates cannot be shared, we need to resort to privacy enhancing technologies such as differential privacy [40] (added noise), fully homomorphic encryption [2, 29, 30] (multiply and add on encrypted data), or secure multi-party computation [63] (joint computation on partial intermediates).

From a practical standpoint of enterprise federated ML, there are use cases for all these means of communication and suitable system infrastructure should support them.

Exploratory and Deployed Pipelines: Enterprise federated ML subsumes both the exploratory ML pipeline development, and the deployment of resulting ML pipelines and models. For exploratory analysis, federated data is repeatably accessed with different pipelines or iterative algorithms. However, federated learning also applies to deployed ML pipelines that process streams of input data. Examples are monitoring and alerting in our fertilizer and paper production use cases. In general, there are different deployment types that differ in federated or central scoring, as well as federated or central usage of scores. Depending on the type and federated data, the necessary parts of a trained model are pushed down to the federated sites. We further distinguish online and offline (semi-manual) data and model exchange, where we focus on infrastructure for online federated learning.

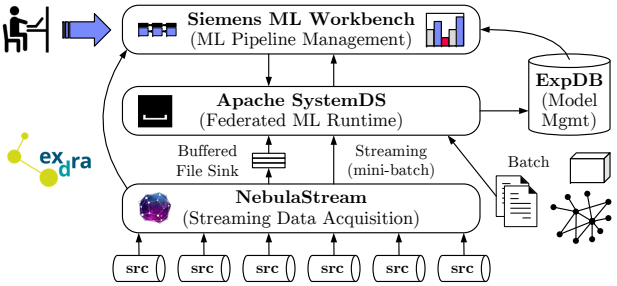

Figure 1: ExDRa System Architecture.

3 SYSTEM ARCHITECTURE

For supporting these federated ML use cases, the ExDRa infrastructure comprises multiple complementary components. Figure 1 shows the system infrastructure. Users interact with a graphical user interface (UI) of an ML workbench for managing data sources and ML pipelines, and exploring results in an interactive manner. Apache SystemDS [8] is extended for executing these ML pipelines on federated raw data. Trained models and their provenance are stored in an ExperimentDB for model management. Besides batch model training on raw data, NebulaStream (NES) [98] further enables streaming data acquisition from IoT devices for both exploratory and deployed pipelines. In this section, we summarize the individual components and their interplay.

3.1 Workbench and Pipeline Management

The UI is part of an ML workbench for project and pipeline management, developed by Siemens. Data scientists create and share projects, which contain data sources, ML pipelines, and interactive result plots. This pipeline management supports a variety of ML systems (e.g., Scikit-learn [68], TensorFlow [1], Apache MXNet [15], and pandas [91]), and the system is designed for good scalability with increasing number of pipelines and users.

Frontend and Backend: The frontend is based on the web framework Angular, and interfaces with a SpringBoot Java backend. Users compose ML pipelines of pre-defined data sources and pipeline steps (e.g., for pre-processing and ML algorithms), through the graphical UI. The backend stores configurations of data sources, pipeline steps, and pipelines. Supported data sources—which can be shared among pipelines—include text files like CSV, hierarchical binary data like HDF5, relational databases, object stores like AWS S3, and even message queues. Similarly, our ExperimentDB (for model analysis) and NES file sinks are integrated as data sources as well. The backend then triggers the execution of pipelines in so-called AI containers, and returns the results to the frontend.

AI Containers: The ML pipelines are executed in Docker containers on AWS ECS or Kubernetes [11]. Each pipeline step is a Python script, and a dedicated container-local Java application (invisible to a user) connects to the central data sources, orchestrates the Python scripts according to the pipeline definition, and returns the results. Similar to the other ML systems, SystemDS and ExperimentDB are used through their respective Python APIs.

3.2 Federated ML Runtime

In this infrastructure, SystemDS is used as federated runtime backend for ML algorithms and pipelines on raw data. For a seamless integration with common data science workflows and the ML workbench (Section 3.1), SystemDS has been extended by a new Python API with lazy evaluation. Users create matrices or frames from federated configurations, files, NumPy arrays, or pandas data frames. The API further exposes operations and higher-level built-in functions, whose calls are collected in a DAG of operations.

```
features = Federated(sds, [node1,node2], ([. . . ],[. . . ]))
model = features.l2svm(labels).compute()
```

On calling `compute`—similar to Dask [73]—on an intermediate, we generate a DML script via depth-first DAG traversal for ordering according to data dependencies, execute this script, and return the result as a NumPy array or data frame. For federated learning, SystemDS supports both federated linear algebra, and federated parameter servers, which we discuss in detail in Section 4. Instead of providing labor-intensive implementations of individual federated ML algorithms, this design aims to provide the necessary primitives such that a wide variety of built-in functions and ML algorithms can be automatically compiled into federated runtime plans.

3.3 Model Management and Experiments

Our ExperimentDB provides means of model and experiment management for exploratory data science. To this end, ExperimentDB comprises two major components: a model and metric store, and a pipeline recommendation engine.

Model Store: We store trained models of pipeline versions (i.e., different artifacts), and their runs (i.e., with different parameters and input data). The inputs are the pipelines of AI containers (Section 3.1), their output models, related accuracy metrics, and other metadata such as lineage and reusable intermediates. If a pipeline is marked for tracking, the workbench backend uses an ExperimentDB API to make these inputs available. The Python scripts are then parsed into an intermediate representation of a data flow graph, and individual pipeline steps are categorized accordingly. We use high-level operator types such as ensembles, estimators, imputers, scalers, selectors, generators, samplers, and transformers. This model store and collected metadata then allows for query-based pipeline comparisons, explanations, and analysis.

Pipeline Recommendation: Beyond basic model management, we further aim to provide (in the future) pipeline recommendations. Given a high-level ML task, dataset and its data characteristics, optional evaluation metric, and history of pipeline runs and their accuracy, the goal is to recommend a ranked list of pipelines for exploration. In contrast to AutoML, this process entails human-machine interaction for pipeline synthesis and debugging, as well as multi-model data analysis, which partially allow for self-supervision and thus, avoids the need for predefined metrics or labels. Our current prototype computes embeddings of pipeline metadata, and trains an ML model to predict scores of pipeline candidates.

3.4 Streaming Data Acquisition

Highly-distributed, potentially moving and unreliable, streaming data sources are a particularly challenging form of federated raw data. Exploratory data science and ML model training are repetitive and thus, require multiple passes over the data. We mitigate this impedance mismatch by leveraging NebulaStream (NES) for managing the low-level aspects of streaming data acquisition.

NES Overview: A central coordinator deploys continuous queries in a decentralized topology of heterogeneous nodes and devices. Inputs are sensor or consolidated logical streams. Operators of physical execution plans are then assigned and re-optimized [32] according to existing queries, available resources, utilization, and topology changes (e.g., moving robots/equipment). The node runtime receives and sends input and output streams, processes multi-threaded tasks, checks privacy constraints, triggers actuators, and provides inbound adapters like OPC (open platform communications) for integrating existing measurement systems.

ExDRa Integration: Within the ExDRa system infrastructure, separate NES instances—each with a coordinator and decentralized topology—get deployed at the individual federated sites, which protects private data by avoiding consolidation in central cloud environments. For exploratory use cases, NES appends the collected streams to file sinks with retention periods (e.g., last two days). ML pipelines then read this federated data from the file sink, and use an in-memory snapshot for iterative training. The file sinks and NES queries are deployed as federated data sources in the ML workbench (Section 3.1). For deployment use cases, NES can also directly feed into the interactive visualization, and trigger mini-batch model updates for online learning. In the future, NES also aims to push down sink predicates, and other operations of deployed ML pipelines, into the continuous queries.

4 FEDERATED RUNTIME

Apache SystemDS [8] supports multiple runtime backends for local, in-memory operations (CPU/GPU), and distributed operations on Apache Spark [97]. The new federated backend[2] follows a similar design of specific federated data objects, and runtime instructions. Here, we describe the overall design, federated linear algebra operations, federated parameter servers, federated data preparation, and our vision for federated ML pipelines on raw data.

4.1 Federated Data and Backend Design

SystemDS compiles user scripts of ML algorithms or pipelines into hybrid runtime programs of local and distributed operations. A main control program (CP)—potentially in the Spark driver—then executes the control flow and operations as instructions. Live variables are accessible via a symbol table. Figure 2 shows this setup and the integration of the federated backend.

[2]The entire federated runtime backend is available open source as part of Apache SystemDS: https://github.com/apache/systemds.

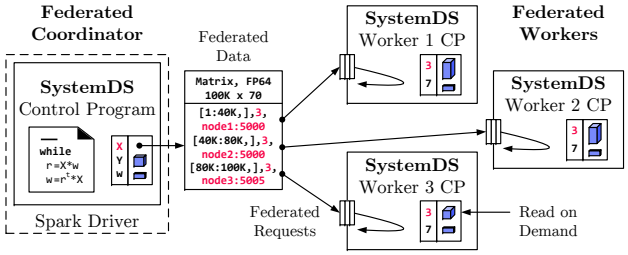

Figure 2: Federated Runtime Backend.

Federated Data: Any non-scalar data objects such as tensors, matrices, and frames can be federated. The main control program acts as the coordinator and holds only metadata—in the form of a federation map—of such federated data. This map stores the data type, value type, dimensions, sparsity, as well as non-overlapping data ranges, and their locations (host, port, data identifier). For example, in Figure 2, we have a federated 100K×70 matrix $\mathbf{X}$, with row partitions $[1:40K]$, $[40K:80K]$, and $[80K:100K]$ on node1, node2, and node3, respectively. If a local operation at the coordinator tries to pin $\mathbf{X}$ into memory, the federated data is transparently transferred—unless it violates coarse-grained or fine-grained privacy constraints—and consolidated in a local matrix.

Federated Workers: Similar to the coordinator, the federated workers are also control programs, but started as worker processes that act like servers at the federated sites. A worker listens on an input queue for incoming RPC requests (called federated requests), executes these requests, maintains a local symbol table, checks privacy constraints (e.g., for data exchange), and finally returns an—optionally SSL-encrypted—RPC response. This design of worker CP programs provides very good flexibility and reuses the I/O subsystem, buffer pool mechanisms, as well as local and distributed operations. For example, it even allows data center federation [89], where a single federated operation triggers distributed operations in a Spark [97] or Flink [4] cluster at the federated site.

Federated Requests: The coordinator communicates with workers through federated requests, using Netty as a network I/O framework for RPCs and data transfers. To simplify the implementation of federated operations, we restricted the federation protocol to only six generic request types:

- `READ(ID, fname)`: Creates a data object from a filename, reads it, and adds it by ID to the symbol table.
- `PUT(ID, data)`: Receives a transferred data object, and adds it by given ID to the symbol table.
- `GET(ID)`: Obtains a data object from the federated site's symbol table, and returns it to the coordinator.
- `EXEC_INST(inst)`: Executes an instruction, which accesses inputs and outputs by IDs in the symbol table.
- `EXEC_UDF(udf)`: Receives a serialized, user-defined function (UDF) object, executes this UDF over requested inputs by ID, may add outputs to the symbol table, and returns a custom object to the coordinator.
- `CLEAR`: Cleans up execution contexts and variables.

For efficiency, the coordinator sends RPCs to all workers in parallel, and a single RPC can contain a sequence of requests and returns a single response. The simplicity of these request types has two profound advantages. First, we can reuse existing instructions for composing federated operations. Second, this design allows for federation hierarchies. If the worker-local data is federated data, a worker can also act as a coordinator of a subgroup of workers.

4.2 Federated Linear Algebra

For broad applicability in various ML algorithms and data science lifecycle tasks, our federated runtime supports both federated linear algebra and federated parameter servers. Federated linear algebra utilizes similar strategies as distributed, data-parallel operations but retains the raw federated data at its federated site. This requirement creates additional challenges and needs compiler support for finding valid yet efficient runtime plans if operations do not directly apply.

Basic Linear Algebra: During compilation and runtime, we check if any inputs are federated data, and dispatch this call to supported federated instructions. Similar to RDD transformations and actions [97], these federated instructions then utilize federated requests—and related high-level primitives for broadcasting and aggregation—to compute the operations over federated data. If no aggregation is needed, the output is itself federated data.

Example 2 (Federated Matrix Multiplication). *Assume a matrix-vector (or matrix-matrix with small right-hand-side) multiplication* $\mathbf{X}\,\mathbf{v}$ *and vector-matrix multiplication* $\mathbf{v}^\top\,\mathbf{X}$ *with* $nrow(\mathbf{X}) \gg ncol(\mathbf{X})$ *and* $\mathbf{X}$ *being composed of federated row partitions. For matrix-vector, we broadcast* $\mathbf{v}$ *via* `PUT`*, execute a local matrix-vector multiplication per partition via* `EXEC_INST`*, which yields a new federated vector with related federation map (logical* `rbind` *here), and finally execute an optional* `rmvar` *instruction via* `EXEC_INST` *to clean up the broadcast* $\mathbf{v}$*. In contrast, for a vector-matrix, we perform a sliced broadcast of* $\mathbf{v}$ *(vector parts according to row ranges), execute a local vector-matrix multiplication per partition via* `EXEC_INST`*, obtain the results via* `GET`*, do a final aggregation via element-wise vector addition at the coordinator, and again, clean up all intermediates.*

Supported Operations: So far, we support—as summarized in Table 1—federated matrix multiplication, unary aggregates, unary element-wise operations, binary matrix-matrix, matrix-vector, and matrix-scalar operations, ternary, quaternary, and parameterized builtin operations, and various reorganizations. These operations further support both row- and column-partitioned federated data via specialized implementations. Most of the binary operations

Table 1: Example Federated Instructions.

Operation Type	Examples
Matmult	`mm`, `tsmm`, `mmchain`
Aggregates	`sum`, `min`, `max`, `sd`, `var`, `mean`
	`rowSums`, ..., `rowMeans`, `colSums`, ..., `colMeans`
Unary	`abs`, `cos`, `exp`, `floor`, `isNA`, `log`, `!`, `round`, `sin`,
	`sign`, `softmax`, `sqrt`, `tan`, `sigmoid`,
Binary	`&`, `cov`, `cm`, `/`, `=`, `>`, `>=`, `%/%`, `<`, `<=`, `log`,
	`max`, `min`, `max`, `-`, `%%`, `*`, `!=`, `|`, `+`, `^`, `xor`
Ternary	`ctable`, `ifelse`, `+*`, `-*`
Quaternary	`wcemm`, `wdivmm`, `wsigmoid`, `wsloss`
Transform/Reorg	`tfencode`, `tfapply`, `tfdecode`,
	`rbind`, `cbind`, `t` (transpose), `removeEmpty`
	`replace`, `reshape`, `X[:,:]` (matrix indexing)

(e.g., matmult, element-wise) support a single federated input and consolidate a second federated input (e.g., aggregated intermediates) in the coordinator. However, whenever two federated inputs are co-partitioned (e.g., because one originated from the other), we directly execute federated operations on them as well.

Higher-level Primitives: SystemDS follows the premise that many data science lifecycle tasks—like data validation, data cleaning, feature and model selection, and model debugging—are themselves based on machine learning and numerical computation [8]. These higher-level primitives are hierarchically composed from built-in functions that rely on linear algebra and thus, are directly supported on federated data as well. In case a binary or ternary operation is not supported over multiple federated matrices, some of them are consolidated in the coordinator, or a privacy exception is thrown if this consolidation would reveal private raw data. For this reason, we are working toward better compiler support that proactively considers privacy constraints and generates efficient runtime plans that adhere to these constraints.

Example 3 (Federated K-Means). *Starting bottom-up, individual ML algorithms are good examples of such higher-level primitives. For instance, consider the inner loop (after initialization and inside a loop for multiple runs) of K-Means clustering, where* $\mathbf{X}$ *is a federated, row-partitioned matrix, and* $\mathbf{C}$ *are the current centroids:*

```
while (term_code == 0) {
  # Compute Euclidean squared distances records-centroids
  D = -2 * (X %*% t(C)) + t(rowSums(C ^ 2));
  # Find the closest centroid for each record
  P = (D <= rowMins(D));
  # If records belong to multiple centroids, share them
  P = P / rowSums(P);
  # Compute the column normalization factor for P
  P_denom = colSums(P);
  # Compute new centroids as weighted averages
  C_new = (t(P) %*% X) / t(P_denom); # ...
}
```

The first matrix multiplication $\mathbf{X}\,\mathbf{C}^\top$ *yields another federated, row-partitioned matrix. The subsequent row aggregates and element-wise operations similarly create aligned federated intermediates, which are then aggregated and only consolidated in aggregate form via colSums*$(\mathbf{P})$ *and* $\mathbf{P}^\top\mathbf{X}$*, where the latter is an aligned matrix multiplication of two federated matrices. Note that this built-in function script is agnostic of local, distributed, or federated input matrices.*

While some ML algorithms directly map to federated operations that preserve private data, other algorithms need dedicated compiler assistance for generating valid runtime plans. Specifying data exchange (i.e., privacy) constraints for federated raw data, tracking derived properties of intermediates and data transfers, and generating constraint-aware plans is an important direction for future work but beyond the scope of this paper.

4.3 Federated Parameter Server

The federated linear algebra backend described so far supports traditional batch ML algorithms (e.g., first- and second-order) as well as high-level primitives, but would be very inefficient for mini-batch algorithms—as used for DNNs—because most federated workers would stay idle while processing a batch at-a-time. Such mini-batch algorithms are usually trained with data-parallel parameter servers or similar distribution strategies. Accordingly, we extended SystemDS' parameter server (PS) for federated data as well.

Background SystemDS PS: The existing SystemDS PS infrastructure implements a traditional, data-parallel parameter server [20, 41, 53, 79], where a central server maintains the current model, and workers pull the model, perform mini-batch iterations over disjoint data partitions to compute gradients, and push these back to the server for aggregation. This functionality is integrated as a native built-in function and can be invoked in a stateless manner:

```
M = list(W1, W2, W3, W4, b1, b2, b3, b4, ... vW4); # model
params = list(lr=lr, mu=mu, stride=stride, pad=pad, ...);
Mp = paramserv(model=M, features=X, labels=y,
               upd=updateGrad, agg=updateModel, utype=ASP,
               freq=BATCH, epochs=200, batchsize=64, ...);
```

The initialized model—as a list of weight and bias matrices, as well as additional state matrices like the parameter velocities—and data is passed as arguments, along with a configuration of update types (e.g., BSP – bulk-synchronous parallel, ASP – asynchronous), update frequencies, batch size, number of epochs, data partitioner, hyper-parameters, and user-defined functions for computing gradients (`updateGrad`), updating the model with gradients (`updateModel`), and optional validation. Internally, the parameter server then either runs in local, multi-threaded or distributed Spark mode, where the latter runs the parameter server locally at the coordinator, spawns a Spark job with standing worker tasks and directly communicates via RPCs between the parameter server and workers.

System Architecture: The federated parameter server exhibits a similar PS architecture, consisting of a server running at the coordinator, and workers at the federated sites for computing gradients on the private data. During setup, we serialize the gradient and update functions and send them to the workers. Then one server thread per worker handles the communication and synchronization with federated workers, which are executed via a `EXEC_UDF(udf)` federated request. Depending on the updated frequency, the model is updated at the worker, and after a fixed number of batches, the accrued gradients are sent to the server for aggregation. In contrast to SystemDS' multi-threaded parameter server, the gradient function is compiled in a way that leverages multi-threaded operations with the available degree of parallelism at each worker.

Federated Data Partitioning: In a typical PS architecture, the input data is potentially shuffled for randomization, and then evenly partitioned horizontally among workers. In contrast, the federated parameter server respects the locality of federated data (specifically, row-partitioned and aligned federated data $\mathbf{X}$ and labels $\mathbf{y}$), which allows only local shuffling and replication of the private federated data. The chosen partitioning strategy—unless the data is used without modification—is again executed via a `EXEC_UDF(udf)` federated request before the start of iterative training.

Handling Imbalance and Skew: A major challenge in the federated setting is the handling of imbalance in terms of different worker data sizes, and skew of the data distributions at the different federated workers. Imbalance and skew (also known as statistical heterogeneity [45, 54]) are problematic because an equal number of epochs results in different numbers of iterations and thus, skew

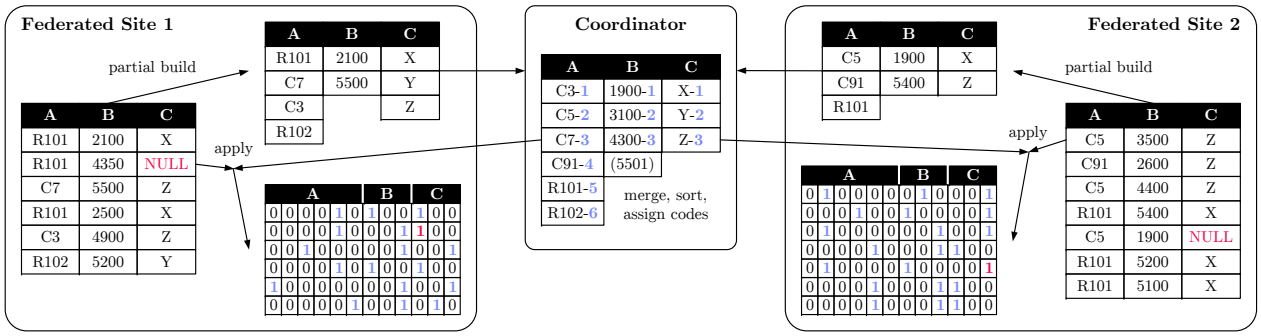

Figure 3: Example Federated Transformencode (binning, recoding, and one-hot encoding).

in execution time, coordination problems in synchronous update strategies like BSP (where the parameter server waits for all worker updates), and biased updates, where the biggest federated data partition solely influences model updates at the end of training. Subsampling large or replicating small partitions on the other hand puts disproportionate weight on the observations of small federated partitions. We are currently using replication with adjusted weights, but subsampling and replication, weighting schemes, and hierarchical parameter server architectures (similar to virtual nodes in Dynamo [21]) is a rich area for future work.

4.4 Federated Data Preparation

Besides ML-based data cleaning and data pre-processing—which are based on federated linear algebra—there are other data preparation techniques that require special federated support. These operations include feature transformations, and data access methods for raw data. In contrast to typically stateless ML systems and libraries, the architecture of standing federated workers further provides rich opportunities for reuse and adaptive data reorganization across multiple pipeline runs of a single user and multiple tenants.

Feature Transformations: Similar to other, commonly-used ML systems, SystemDS supports—through operations like frame `transformencode` and `transformapply`—standard feature transformations like recoding (categories to integers), feature hashing (categories to upper-bounded integers, potentially with collisions), binning (numeric values to integers), and one-hot encoding (integers to sparse boolean vectors). The federated instructions of these operations leverage the flexibility of UDFs via `EXEC_UDF` and preserve privacy of the raw federated data. In detail, federated `transformencode` uses a two-pass approach. First, we build encoder-specific metadata for all non-pass-through features (i.e., all columns except unmodified numeric columns) at the federated sites, as well as consolidate—and optionally sort—the metadata for consistent encoding. Second, we broadcast the aggregated metadata, and in a second pass over the federated data, then perform the actual encoding. The outputs are a federated encoded matrix with consistently aligned one-hot encoded features (equivalent to local encoding), and a local metadata frame.

Example 4 (Federated One-hot Encoding). *Figure 3 shows an example of a row-partitioned federated input frame with three columns A, B, and C. Here, A and C are categorical features subject to recoding and one-hot encoding, while B is a numerical feature subject to binning (with three equi-width bins) and one-hot encoding. Not all categories will appear in all federated sites. For example, feature A represents regular and custom recipes of our paper production use case from Section 2.2. In a first step, the federated workers compute the distinct items for recoded features and min/max values for binned features. The coordinator consolidates these metadata frames from the federated sites, merges and sorts the distinct items, computes the bin boundaries, and finally assigns contiguous integer codes. In a second step, the global metadata is sent to the federated sites for encoding the input frame into a numeric matrix. Non-existing categories lead to all-zero columns but this encoding ensures consistent feature positions for model training. Federated linear algebra then further allows applying various techniques for data cleaning and missing value imputation. These primitives are implemented as script-based built-in functions and thus, can now be compiled into federated runtime plans as well. For instance, the NULLs (or NAs) in column C might be imputed with the mode (most frequent categorical value, imputing* X *or* Z*), via robust functional dependencies [22] (assuming or discovering A → C, and imputing* X *and* Z*, respectively), or by more advanced, ML-based techniques like multivariate imputation by chained equations (MICE, which trains regression and classification models for imputing categorical and numerical columns) [13, 86].*

Improved Feature Transformations: There are many opportunities for improved federated feature transformations. First, techniques like zigzag joins [84]—that rely on Bloom filters for pre-filtering—can be adapted for determining categories that need to be exchanged with the coordinator, thereby reducing data transfer and revealed information. Similarly, for features that only exist at a single federated site (e.g., column-partitioned federated data), we only need to exchange the number (instead of the set) of distinct items. Second, there is a tradeoff between privacy and performance versus accuracy. Instead of recoding, users can resort to feature hashing (with an agreed hash function), which is computed in a purely federated manner without data exchange. However, hash collisions merge multiple categories into one feature, which might negatively affect accuracy. In our current implementation we support the different transformations but leave the choice up to the user because we expect related negotiations among involved parties.

ML Pipelines on Raw Data: The federated raw data is read as frames or heterogeneous tensors [8] into the standing workers and then accessed with federated indexing (e.g., for feature selection) and feature transformations (e.g., `transformencode`). Similar to low-latency scoring services, we already reuse deserialized recode

maps at a fine granularity of individual features. Inspired by query processing on raw data [3, 35, 47], data reorganization via database cracking [36] / adaptive indexing [37], recycling of intermediates [38], and lazy maintenance of materialized views [101], we further aim to leverage the opportunity of standing workers for adaptive data reorganization. Future work includes three main directions.

- *Lineage-based Reuse:* First, we will establish lineage-based caches for reuse [23, 93] and debugging [87] of intermediates. As a first step, we have already integrated our LIMA framework for fine-grained lineage tracing and reuse [69]. In this context, there is also potential for multi-tenant data structures that share partial overlap of feature subsets.
- *Compression:* Second, free cycles of federated workers can be used for asynchronous, lossless compression [25, 52] such as compression planning and compaction of intermediates.
- *Incremental Maintenance:* Third, the cached and reorganized intermediates can be—in case of applicable operations—incrementally maintained [66, 75] for new or deleted data.

In the context of exploratory ML pipelines with repeated raw data access and data enrichment, such a systematic reorganization would allow to eliminate unnecessary redundancy, and specialize the data representation for the observed workload characteristics, while retaining the appearance of a stateless ML system and preserving the privacy of the federated raw input data.

5 TOWARDS DEPLOYMENT IN PRODUCTION

The ExDRa infrastructure is *not* deployed in production yet. However, the use cases described in Section 2 are real production use cases, which exhibit potential for enterprise federated learning. In this section, we discuss in more detail the envisioned deployment of the ExDRa infrastructure, and other deployment models.

5.1 Envisioned Deployment: Private Data

SystemDS and NebulaStream are both large system projects, developed independently for different use cases of the end-to-end data science lifecycle and streaming IoT data management. For enterprise federated learning in the ExDRa infrastructure, both systems are jointly extended and their strengths come to bear in the envisioned deployment model. Most importantly, this setup enables us to mitigate the impedance mismatch between streaming data sources and iterative, multi-pass federated learning.

Basic Deployment: Figure 4 shows the basic deployment model of enterprise federated ML in ExDRa. We assume a moderate number of federated sites (recently called cross-silo federated learning [45]), each with their local infrastructure and private raw data. At each federated site, a SystemDS worker is started as a standing server process, receiving federated requests from the coordinator via secure communication channels, and accessing permissioned raw data. Each site also deploys a NES instance with a coordinator and decentralized topology of workers and sensors for streaming data acquisition through distributed continuous queries (which are pre-configured but there is potential for partial push-down of deployed scoring pipelines). For training, NES appends the collected data to file sinks with configured retention periods, and every SystemDS ML training session reads the data, and works on a consistent in-memory snapshot. Thus, batch data and buffered

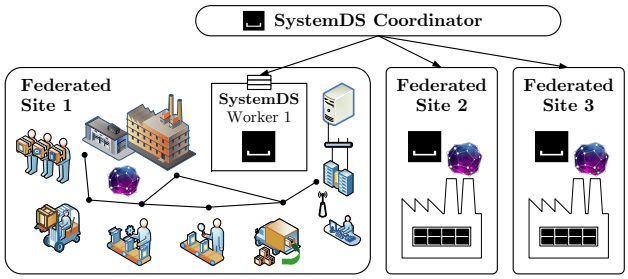

Figure 4: Envisioned SystemDS and NES Deployment.

streams are handled the same, which bridges the gap to iterative ML training. This execution model has well-defined semantics but long training sessions might work on partially stale data.

Extensions for Stream Ingestion: In contrast to the generic basic deployment, there are more specialized scenarios that allow ingesting input streams into SystemDS workers as well. First, federated parameter servers (see Section 4.3) process the data in epochs, one batch of rows at a time. Here, the federated workers can seamlessly handle the removal or append of new batches according to the configured retention periods. However, changing data sizes require coordination to obtain imbalance ratios for replication and weight adjustments. Second, for deployed scoring pipelines SystemDS can act as a serving platform, processing the input streams and returning the predictions to the federated site or coordinator.

The Missing Pieces: For deployment in production several challenges remain. First, from an organizational perspective, appropriate legal frameworks and business models need to be devised. Enterprise federated learning can be a win-win for large and small companies alike: large companies might run the infrastructure and model training; small and medium-sized enterprise (SMEs) may participate as federated sites, which opens up new income channels. In contrast to existing data marketplaces [28], participating federated sites can contribute to global model training without sharing their precious raw data. A crucial aspect is finding the right incentives for motivating the federated sites to participate with high-quality raw data (e.g., Jordan's leave-one-out proposal [42], or contractual benefits). Second, from a technical perspective, deployment in production requires a trusted system infrastructure, and techniques for generating and verifying federated execution plans under awareness of privacy constraints. Third, it is a never-ending quest for added business value, stronger privacy, and higher performance.

5.2 Other Deployments: Private Models

Beyond our envisioned deployment for private federated data, there are other interesting deployments where federated ML models instead of federated data are kept private. For example, consider a scenario of a subsidiary company that receives—potentially anonymized—data from the parent company in order to derive predictions or create insights. Here, the ML models are the subsidiary's core assets. Similar examples include telecom providers sharing mobility data with partners for deriving insights, and hedge funds sharing financial data and predicting the stock market based on a stake-weighted federated ensemble of private models [17]. Like enterprise federated ML, sharing only predictions prevents reverse-engineering (reconstruction) of the underlying private models.

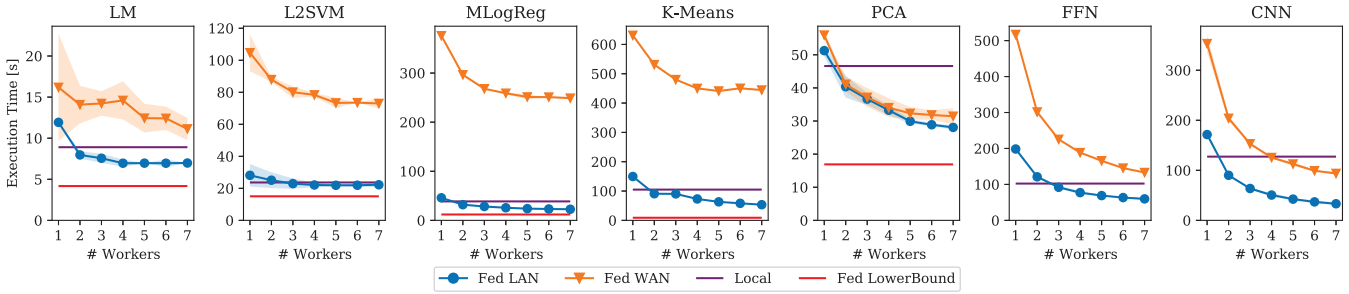

Figure 5: Basic Algorithm Comparison and Scalability with # Federated Workers ($10^6 \times 1{,}050$ feature matrix X).

6 EXPERIMENTS

Our experiments study the performance of the described federated runtime backend of SystemDS (Section 4), with various ML algorithms and pipelines, network configurations, and in comparison with local execution and other ML systems.

6.1 Experimental Setup

Baselines: For evaluating the characteristics of federated linear algebra and parameter servers in controlled yet practically relevant scenarios, we compare the following four main baselines:

- *Local:* Our main baseline is SystemDS with local, in-memory operations, which uses equivalent runtime plans and runtime operations, but executed locally on a single node.
- *Federated LAN:* The federated runtime backend dispatches runtime operations on federated matrices to the described federated linear algebra operations and parameter server. For Federated LAN, all coordinator and workers nodes are part of a local area network (LAN) of two racks, connected via an HPE FlexFabric5710 48XGT switch.
- *Federated WAN:* In addition to Federated LAN, we experiment with the federated backend in a wide-area network (WAN) setting. Here, a client node runs the coordinator in Copenhagen, Denmark and the workers run in a cluster (described below) in Graz, Austria – a distance of more than 1,000 km with round-trip latency of about 35-60 ms, and data transfer bandwidth of about 1.4-2 MB/s.
- *Other ML Systems:* To ensure that *Local* is a competitive baseline, we also compare with local execution in Scikit-learn 0.23 [68] for traditional batch ML algorithms, and TensorFlow 2.3.1 [1] for mini-batch neural network workloads. These systems do not support federated ML.

Cluster Configuration: We ran all experiments described here on eight nodes, each having a single AMD EPYC 7302 CPU at 3.0–3.3 GHz (16 physical/32 virtual cores), 128 GB DDR4 RAM at 2.933 GHz balanced across 8 memory channels, 2×480 GB SATA SDDs (system), 12×2 TB SATA HDDs (data), and 2×10Gb Ethernet. The nominal peak performance of each node is 768 GFLOP/s and 183.2 GB/s, whereas we measured 109.6 GB/s for an 8 GB matrix-vector multiplication. For wide-area network tests, we use an additional client node Dell XPS 15 with one Intel i9-9980HK CPU at 2.4–5.0 GHz (8 physical/16 virtual cores), and 32 GB DDR4 RAM at 2.666 GHz. Our software stack comprises Ubuntu 20.04.1 as operating system, OpenJDK Java 1.8.0_265, and SystemDS 2.0.0++ (as of 03/2021), configured with native Intel MKL BLAS for dense matrix-matrix multiplications. The coordinator and worker nodes use consistent JVM configurations of `-Xmx110g -Xms110g -Xmn11g`, while the WAN client uses `-Xmx30g -Xms30g -Xmn3g`.

Workloads: The tested workloads include the ML algorithms linear regression (LM), L_2-regularized support vector machine (L2SVM) and multi-class logistic regression (MLogReg) for classification, K-Means for clustering (with K=50 centroids), principal component analysis (PCA) for dimensionality reduction (with K=10 projected features), as well as two parameter server models: a fully-connected feed-forward network (FFN) with BSP, 5 epochs, batch size 512, and trained with stochastic gradient decent (SGD) with Nesterov momentum, as well as a convolutional neural network (CNN) with BSP, 2 epochs, batch size 128, and standard SGD. These algorithms are trained on a synthetic $1\text{M} \times 1{,}050$ feature matrix (after one-hot encoding categorical features), which closely resembles the characteristics of the data from our paper production use case described in Section 2.2. For the CNN scenario though, we use the standard $60\text{K}/10\text{K} \times 784$ MNIST dataset from computer vision. The feature matrix $\mathbf{X}$ is stored as a row-partitioned, federated matrix with balanced partition sizes at the federated sites (i.e., worker nodes), while the labels $\mathbf{y}$ are stored at the coordinator node. We fix the number of maximum iterations for iterative ML algorithms and report the end-to-end runtime—including JVM startup and I/O from binary files—as a mean of (at least) three repetitions.

6.2 ML Algorithms Performance

In a first set of experiments, we compare the local ML algorithms performance with both Federated LAN and WAN. We also vary the number of federated workers, evaluate communication settings such as SSL encryption, and compare other ML systems.

ML Algorithms: Figure 5 shows the end-to-end runtime of the ML algorithms. As a first step, consider a scenario of three federated workers (the number of workers is varied on the x-axis), which require additional communication but also provide more computational resources. The ML algorithms have different characteristics in that regard. First, LM internally calls an iterative conjugate-gradient LM method (used for $\text{ncol}(\mathbf{X}) > 1{,}024$), where each iteration performs an $\mathbf{X}^\top(\mathbf{X}\,\mathbf{v})$ over the federated data. Compared to local, we observe low overhead and already a runtime improvement with three workers. The Fed LowerBound represents the remaining local execution time that is not subject to federated computation and thus, the best Fed LAN could achieve. Second, L2SVM uses two nested while loops, where each outer iteration computes gradients, and

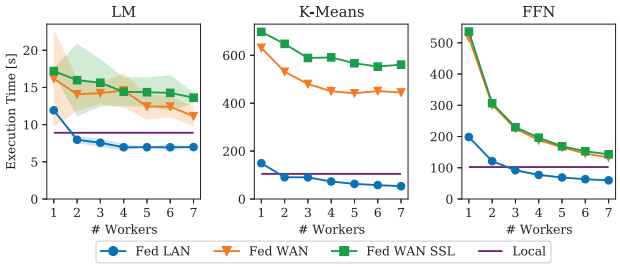

Figure 6: Comparison of Communication Settings.

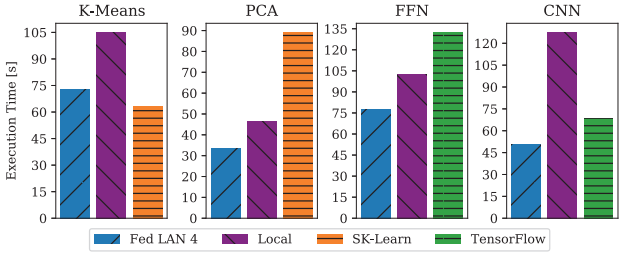

Figure 7: Comparison with Other ML Systems.

the inner loop performs a line search along the gradient. Since the federated $\mathbf{X}$ is only accessed via matrix-vector and vector-matrix operations in the outer loop, the differences to the local runtimes are much smaller. Third, MLogReg also uses two nested while loops, but each inner iterations performs an $\mathbf{X}^\top(\mathbf{w} \odot (\mathbf{X}\,\mathbf{v}))$ on the federated $\mathbf{X}$ and accordingly, we see again a solid improvement with three workers. Fourth, a single run of K-Means has a single while loop, which uses more compute-intensive matrix-matrix multiplications as shown in Example 3. Fifth, PCA is a non-iterative algorithm and computes an Eigen decomposition of $\mathbf{X}^\top\mathbf{X}$ and subsequently, projects the data via another matrix multiplication to K=10 features. With large number of rows, the two matrix multiplications dominate the runtime. Both K-Means and PCA accordingly show also substantial improvements compared to local execution. Finally, FNN and CNN use the mini-batch parameter server architecture with local per-batch updates and global per-epoch synchronization. The larger compute resources of the federated backend yield improvements despite the additional communication. Most importantly, none of these federated ML algorithms ever communicates the raw input data to the coordinator (and thus, preserve privacy of the federated data), they all show only small overhead, and in many cases even runtime improvements. In additional experiments with federated labels $\mathbf{y}$ and/or smaller number of columns (not shown here), we observe that some algorithms like L2SVM incur substantially larger overhead though, because all vector operations of the inner loop are then converted to federated operations as well, which increases communication without benefiting from the larger computational resources. In the Federated WAN setting, the relative communication overhead is also substantially higher, but even there, the end-to-end overhead is moderate, which renders federated learning practical for real deployments.

Scalability: Besides the comparison of Local, Federated LAN, and Federated WAN, Figure 5 also shows the scalability of our federated backend with increasing number of federated workers. We investigate strong scaling behavior by keeping the data size constant. The coordinator sends federated requests in parallel to all the workers and either broadcasts all side inputs or only relevant slices according to operation requirements. The size of communicated intermediates is moderate in all scenarios though; typically, we exchange only vectors in the number of rows (LM, L2SVM, MLogReg, KMeans) or columns (LM, L2SVM) of $\mathbf{X}$, columns-by-classes (MLogReg), columns-by-centroids (K-Means), columns-by-columns (PCA), or model sizes (FFN, CNN). Accordingly, we see good scalability, where additional workers even improve the runtime up until a point, where the partitions per worker become so small that communication increasingly dominates the total runtime. For L2SVM and LM, the improvements are smaller because L2SVM is dominated by vector operations at the coordinator, and LM has a very small runtime, where initial startup constitutes a large fraction of total execution time. In the Federated WAN, the communication overhead is larger but still moderate overall. As the number of workers increases both federated computation and—maybe surprisingly—communication time reduces. The coordinator sends RPCs to all workers in parallel (which mitigates the additional latency), and the more workers the smaller some of the transferred intermediates (e.g., n/#workers).

SSL Encryption: As mentioned in Section 4.1, the federated backend of SystemDS supports SSL-encrypted communication channels between the coordinator and federated workers. We leverage Netty's `SslContext` for encrypting the federated requests and responses including exchanged data. In a next experiment, we study the overhead this encrypted communication entails. Figure 6 compares LM, K-Means, and FFN—which have very different characteristics and thus, showed different scaling behavior—in the Federated LAN, Federated WAN, and Federated WAN with SSL settings. For LM—where exchanged intermediates are small (vectors in the number of columns)—the overhead of WAN and additional SSL encryption is limited to about 2x and 10%, respectively. K-Means shows larger overhead of about 4-8x in a WAN setting due to more iterations and larger transfers (columns-by-centroids), and again about 15% overhead for SSL. In contrast, the federated parameter server shows only moderate WAN and SSL overhead because of the higher computational workload per worker and infrequent per-epoch global model updates and synchronization.

ML System Comparison: With the comparison of local and federated algorithms in mind, we can now turn to a comparison with other ML systems, specifically Scikit-learn [68] and TensorFlow [1] as widely-used ML systems. We select K-Means, PCA, FFN, and CNN for comparison in order to limit the influence of algorithmic differences. The algorithms were configured to yield a similar number of iterations (e.g., K-Means) and final accuracy. Figure 7 shows the results comparing local and Federated LAN configurations of K-Means and PCA with Scikit-learn, and FFN and CNN with TensorFlow. Overall, we observe mixed results. K-Means is 1.6x slower than Scikit-learn, while PCA is 2x faster. Similarly, FFN is 25% faster, while CNN is 2x slower than TensorFlow. We attribute these differences to remaining algorithmic discrepancies, and the comparison with best-of-breed ML systems for the different algorithms, whereas SystemDS aims to support a wide range of algorithms and deployments. For CNN, the overhead is partially due to SystemDS using sparse conv2d_backward data/filter and other operations because MNIST and related intermediates are just below the internal sparsity threshold. Moreover, TensorFlow's parallel

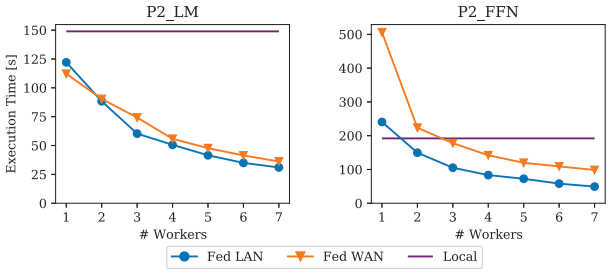

Figure 8: ML Pipeline Scalability with # Federated Workers.

operator scheduling is advantageous in small mini-batch scenarios. In additional experiments on a spectrum of data characteristics, we observed relative improvements of SystemDS compared to the other ML systems with increasing sparsity, number of rows, and batch size. Most importantly, these comparisons ground the observed Federated LAN results in a performance range close to state-of-the-art systems, supporting the conclusion of applicability in practice.

6.3 ML Pipelines Performance

In a second set of experiments, we now return to our main motivation of supporting entire ML pipelines on federated raw input data without central data consolidation. We evaluate ML pipelines of the use cases described in Section 2 on synthetic data.

ML Pipeline Setup: The workload is a simplified training pipeline P2 of the paper production use case (see Section 2.2). This pipeline reads the input data of continuous and categorical features as a federated frame, and transforms the frame via recoding and one-hot encoding (see Section 4.4) into a numeric input matrix and a meta frame that holds the recode maps. Subsequently, we perform value clipping for values outside the interval $[-1.5\sigma, 1.5\sigma]$ of column standard deviations, normalize the data to zero column means and column standard deviations one, and finally, create 70/30 train and test splits. In order to retain a balanced data distribution across federated workers, we perform this splitting via a uniformly sampled selection-matrix-multiply. Finally, we train a regression or neural network model on the train split, evaluate its performance on the test split, and write out the model and metadata.

Scalability: Figure 8 shows the total execution time of ML pipeline P2 on a synthetic federated dataset of 10^6 observations that map—after encoding—to a 1M×1,050 feature matrix. As the number of workers increases, we again see good improvements compared to local operations. The federated transformencode, pre-processing like outlier removal and normalization, train/test splitting, and LM training nicely map to federated linear algebra operations. P2_LM and P2_FNN differ only in the used training algorithm. The larger compute workload of P2_FNN then explains the better scalability with more workers. For P2_LM, already a single worker shows improvements over local execution because of the additional resources of a coordinator and one worker compared to a single node, which can be used for garbage collection and JIT compilation. Finally, we also partially support the remaining ML pipelines of our use cases in a federated environment. These pipelines include pre-processing steps like missing value imputation, PCA, correlation matrices, density-based clustering, as well as the task-parallel training of multiple GMM (Gaussian Mixture Model) instances.

7 RELATED WORK

The ExDRa system is related to existing work of data analysis on raw data, federated learning systems, as well as model and pipeline management. We combine and extend techniques of these areas for exploratory data science workflows on federated raw data.

Data Analysis on Raw Data: Query processing on raw data [3, 35] processes SQL queries directly on raw input files in order to avoid expensive data loading in the context of exploratory analysis. To alleviate repeated parsing and data access overhead, for example, positional maps and attribute caches are transparently build and exploited during repeated analysis of the same files [3]. Furthermore, code generation techniques are used to specialize the query engine for heterogeneous input formats [47]. Orthogonal to existing work, we aim to support data pre-processing pipelines and linear algebra programs on such raw data sources. This research direction is also related to different categories of ML systems. First, there are ML systems with good support for multi-modal input features such as DeepDive [78], Overton [70], and Ludwig [64]; systems for more complex feature extraction, and pre-processing like TFX [7] and SageMaker [18]; and systems for data cleaning like HoloClean [34, 72], AlphaClean [51], BoostClean [50], and CPClean [46]. Second, there are ML systems that exploit materialized intermediates for reuse and debugging such as Columbus [99], Helix [93], Collaborative Optimizer [23], and Mistique [87]. In contrast to these two categories, we provide system infrastructure for executing and optimizing ML pipelines on federated raw data.

Federated Learning: Privacy-preserving ML is a very active research area with major challenges and opportunities. A major direction is the use and specialization of privacy-enhancing technologies such as differential privacy [40], fully homomorphic encryption (FHE) [2, 29, 30], and secure multi-party computation (MPC) [63], where FHE and MPC are often only used for inference, that is, scoring private user input [19, 30, 44]. In contrast, federated learning broadly aims to learn ML models without central data consolidation and exposing raw input data. Early work on federated or decentralized learning [9, 56, 60, 61] adopted an architecture similar to traditional parameter servers [20, 41, 53, 79], but augmented this architecture by additional techniques such as client sampling [60], agent-based, fault tolerant, decentralized aggregation [9, 10], peer-to-peer gradient and model exchange [56, 57, 90], and meta learning for federated, private recommendation models [58, 65]. A major challenge for federated parameter servers is statistical (and hardware) heterogeneity [45, 54] that can negatively affect convergence [12]. Recent techniques include reducing variance [82], selecting relevant subsets of data [85], tolerating partial client work [55], partitioning the client population into independent, congruent groups [74], and adaptive optimization at the server with standard SGD at the clients [71]. For such algorithmic improvements, also new benchmarks such as LEAF [12] are emerging.

Federated Learning Systems: In contrast to algorithmic work on federated learning, infrastructure for federated learning is relatively sparse, and includes agent-based federated (parameter) servers [9], and systems that rely on privacy-enhancing technologies tailored for specific algorithms such as tree-based models [92]. A careful reader might wonder if work on in-DBMS ML [24, 39, 59] could be combined with traditional federated query processing

[43] through wrappers, external tables, or SparkSQL's data sources [5]. While conceptually possible, the wrappers would need to allow the push down of partial linear algebra operations, and the federated sources would also require a linear algebra and parameter server runtime. More relevant to federated learning, TensorFlow Federated (TFF) [31, 45] recently provides high-level primitives such as `tff.federated_broadcast`, `tff.federated_map`, and `tff.federated_mean` for composing federated computation. This TFF API shares design aspects with our federated backend (at a higher level than our federated requests), but is at a much lower level than linear algebra programs, focuses primarily on simulating federated environments, and an open-source federated device runtime does not exist yet. In contrast to such dedicated APIs, we aim to support federated data preparation, parameter servers, and linear algebra programs via automatic plan generation, and thus, a wide variety of existing algorithms and higher-level primitives without the need for users to implement federated learning.

Model and Pipeline Management: The model and pipeline management components in ExDRa are related to several subareas. First, ML pipeline management in Columbus [99], KeystoneML [81], and TuPaQ [80] similarly aim to remove unnecessary redundancy in feature selection and hyper-parameter tuning workloads. Second, recent frameworks like MLflow [14, 96] and ModelDB [88] also provide catalogs and repositories of pipelines and models, experiment tracking, and related provenance information. Third, AutoML tools like Auto-WEKA [83], Auto-WEKA 2.0 [48], Auto-sklearn [26], TPOT [67], Alpine Meadow [77], Amazon SageMaker Autopilot [18], and other AutoML cloud services (e.g., Azure ML, Google AutoML) [100] provide means of pipeline recommendations. Beyond state-of-the-art, ExDRa focuses on data science workflows on federated raw data, and pipeline recommendations that offer good accuracy-runtime tradeoffs in this challenging context.

8 CONCLUSIONS

To summarize, we gave an overview of the ExDRa system for exploratory data science on federated raw data. We described use cases, deployment and privacy models, the system architecture, selected features of SystemDS' federated runtime backend, and promising experimental results. Compared to local ML training, federated linear algebra programs and federated parameter servers exhibit only small to moderate overhead (and can even improve the runtime). At the same time, federated learning preserves the privacy of federated raw data via practical means such as aggregates and encrypted communication. In conclusion, federated learning shows great promise for broad applicability in practice. The automatic generation of federated execution plans then allows reusing and deploying a wide variety of ML algorithms, as well as data preparation and model debugging techniques in such federated environments. However, a deployment in a multi-company context further requires a trusted system infrastructure, and means of generating and verifying federated execution plans under awareness of privacy constraints. Beyond this hardening of the infrastructure, we see many opportunities for future work, especially regarding ML pipelines on federated raw data, a fine-grained spectrum of privacy enhancing technologies with different tradeoffs, and self-balancing incentives for participating federated sites.

ACKNOWLEDGEMENTS

The ExDRa project is funded through the bilateral program "ICT of the Future – Smart Data Economy" by the German Federal Ministry for Economic Affairs and Energy (BMWi, 01MD19002), and the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK, 873838). Additional contributions by Sebastian Benjamin Wrede were made within the Know-Center GmbH, DDAI COMET module (funded by BMK, BMDW, FFG, SFG, and industrial partners). Furthermore, we also thank Arnab Phani, David Weissteiner, and Valentin Leutgeb for their valuable contributions to the implementation of SystemDS' federated backend, as well as our anonymous reviewers for their constructive criticism, comments, and suggestions.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*. 265–283.

[2] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2018. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Comput. Surv.* 51, 4 (2018), 79:1–79:35.

[3] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. 2012. NoDB: Efficient Query Execution on Raw Data Files. In *SIGMOD*. 241–252.

[4] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. 2014. The Stratosphere platform for big data analytics. *VLDB J.* 23, 6 (2014), 939–964.

[5] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In *SIGMOD*. 1383–1394.

[6] Yannis Bakos, Hanna Halaburda, and Christoph Müller-Bloch. 2021. When permissioned blockchains deliver more decentralization than permissionless. *Commun. ACM* 64, 2 (2021), 20–22.

[7] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *SIGKDD*. 1387–1395.

[8] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie N. Lindstaedt, Arnab Phani, Benjamin Rath, Berthold Reinwald, Shafaq Siddiqui, and Sebastian Benjamin Wrede. 2020. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR*.

[9] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *MLSys*.

[10] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *CCS*. 1175–1191.

[11] Brendan Burns, Brian Grant, David Oppenheimer, Eric A. Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes. *Commun. ACM* 59, 5 (2016), 50–57.

[12] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. LEAF: A Benchmark for Federated Settings. *CoRR* abs/1812.01097 (2018).

[13] José Cambronero, John K. Feser, Micah J. Smith, and Samuel Madden. 2017. Query Optimization for Dynamic Imputation. *PVLDB* 10, 11 (2017), 1310–1321.

[14] Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, and Corey Zumar. 2020. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. In *DEEM@SIGMOD*. 5:1–5:4.

[15] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR* abs/1512.01274 (2015).

[16] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, and Caleb Welton. 2009. MAD Skills: New Analysis Practices for Big Data. *PVLDB* 2, 2 (2009), 1481–1492.

[17] Richard Craib, Geoffrey Bradway, Xander Dunnwith, and Joey Krug. 2017. Numeraire: A Cryptographic Token for Coordinating Machine Intelligence and Preventing Overfitting. https://numer.ai/

[18] Piali Das, Nikita Ivkin, Tanya Bansal, Laurence Rouesnel, Philip Gautier, Zohar S. Karnin, Leo Dirac, Lakshmi Ramakrishnan, Andre Perunicic, Iaroslav Shcherbatyi, Wilton Wu, Aida Zolic, Huibin Shen, Amr Ahmed, Fela Winkelmolen, Miroslav Miladinovic, Cédric Archembeau, Alex Tang, Bhaskar Dutt, Patricia Grao, and Kumar Venkateswar. 2020. Amazon SageMaker Autopilot: a white box AutoML solution at scale. In *DEEM@SIGMOD*. 2:1–2:7.

[19] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *PLDI*. 142–156.

[20] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *NeurIPS*. 1232–1240.

[21] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-value Store. In *SOSP*. 205–220.

[22] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System. In *CIDR*.

[23] Behrouz Derakhshan, Alireza Rezaei Mahdiraji, Ziawasch Abedjan, Tilmann Rabl, and Volker Markl. 2020. Optimizing Machine Learning Workloads in Collaborative Environments. In *SIGMOD*. 1701–1716.

[24] Joseph Vinish D'silva, Florestan De Moor, and Bettina Kemme. 2018. AIDA - Abstraction for Advanced In-Database Analytics. *PVLDB* 11, 11 (2018), 1400–1413.

[25] Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. 2018. Compressed linear algebra for large-scale machine learning. *VLDB J.* 27, 5 (2018), 719–744.

[26] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2019. Auto-sklearn: Efficient and Robust Automated Machine Learning. In *Automated Machine Learning*. 113–134.

[27] Food and Agriculture Organization of the United Nations. 2017. *World fertilizer trends and outlook to 2020, Summary Report*. http://www.fao.org/3/a-i6895e.pdf.

[28] Michael Fruhwirth, Michael Rachinger, and Emina Prlja. 2020. Discovering Business Models of Data Marketplaces. In *HICSS*. 1–10.

[29] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *STOC*. 169–178.

[30] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML*, Vol. 48. 201–210.

[31] Google. 2020. TensorFlow Federated: Machine Learning on Decentralized Data . https://www.tensorflow.org/federated

[32] Philipp M. Grulich, Sebastian Breß, Steffen Zeuch, Jonas Traub, Janis von Bleichert, Zongxiong Chen, Tilmann Rabl, and Volker Markl. 2020. Grizzly: Efficient Stream Processing Through Adaptive Query Compilation. In *SIGMOD*. 2487–2503.

[33] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2020. Building High Throughput Permissioned Blockchain Fabrics: Challenges and Opportunities. *PVLDB* 13, 12 (2020), 3441–3444.

[34] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*. 829–846.

[35] Stratos Idreos, Ioannis Alagiannis, Ryan Johnson, and Anastasia Ailamaki. 2011. Here are my Data Files. Here are my Queries. Where are my Results?. In *CIDR*. 57–68.

[36] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007. Database Cracking. In *CIDR*. 68–78.

[37] Stratos Idreos, Stefan Manegold, and Goetz Graefe. 2012. Adaptive indexing in modern database kernels. In *EDBT*. 566–569.

[38] Milena Ivanova, Martin L. Kersten, Niels J. Nes, and Romulo Goncalves. 2010. An architecture for recycling intermediates in a column-store. *ACM Trans. Database Syst.* 35, 4 (2010), 24:1–24:43.

[39] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J. Gao. 2019. Declarative Recursive Computation on an RDBMS. *PVLDB* 12, 7 (2019), 822–835.

[40] Zhanglong Ji, Zachary Chase Lipton, and Charles Elkan. 2014. Differential Privacy and Machine Learning: a Survey and Review. *CoRR* abs/1412.7584 (2014).

[41] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware Distributed Parameter Servers. In *SIGMOD*. 463–478.

[42] Michael Jordan. 2018. SysML: Perspectives and Challenges. https://www.youtube.com/watch?v=4inIBmY8dQI MLSys Keynote.

[43] Vanja Josifovski, Peter M. Schwarz, Laura M. Haas, and Eileen Tien Lin. 2002. Garlic: a new flavor of federated query processing for DB2. In *SIGMOD*. 524–532.

[44] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security Symposium*. 1651–1669.

[45] Peter Kairouz, Brendan McMahan, and Virginia Smith. 2020. Federated Learning Tutorial. In *NeurIPS*. https://slideslive.com/38935813/federated-learning-tutorial

[46] Bojan Karlas, Peng Li, Renzhi Wu, Nezihe, Merve Guerel, Xu Chu, Wentao Wu, and Ce Zhang. 2021. Nearest Neighbor Classifiers over Incomplete Information: From Certain Answers to Certain Predictions. *PVLDB* (2021).

[47] Manos Karpathiotakis, Ioannis Alagiannis, and Anastasia Ailamaki. 2016. Fast Queries Over Heterogeneous Data Through Engine Customization. *PVLDB* 9, 12 (2016), 972–983.

[48] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *J. Mach. Learn. Res.* 18 (2017), 25:1–25:5.

[49] Ahmed Koubaa and Zoltan Koran. 1995. Measure of the internal bond strength of paper/board. *Tappi Journal* 78 (1995).

[50] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, and Eugene Wu. 2017. BoostClean: Automated Error Detection and Repair for Machine Learning. *CoRR* abs/1711.01299 (2017).

[51] Sanjay Krishnan and Eugene Wu. 2019. AlphaClean: Automatic Generation of Data Cleaning Pipelines. *CoRR* abs/1904.11827 (2019).

[52] Fengan Li, Lingjiao Chen, Yijing Zeng, Arun Kumar, Xi Wu, Jeffrey F. Naughton, and Jignesh M. Patel. 2019. Tuple-oriented Compression for Large-scale Mini-batch Stochastic Gradient Descent. In *SIGMOD*. 1517–1534.

[53] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *OSDI*. 583–598.

[54] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* 37, 3 (2020), 50–60.

[55] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *MLSys*.

[56] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. In *NeurIPS*. 5330–5340.

[57] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. 2018. Asynchronous Decentralized Parallel Stochastic Gradient Descent. In *ICML*. 3049–3058.

[58] Yujie Lin, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Dongxiao Yu, Jun Ma, Maarten de Rijke, and Xiuzhen Cheng. 2020. Meta Matrix Factorization for Federated Rating Predictions. In *SIGIR*. 981–990.

[59] Shangyu Luo, Zekai J. Gao, Michael N. Gubanov, Luis Leopoldo Perez, and Christopher M. Jermaine. 2017. Scalable Linear Algebra on a Relational Database System. In *ICDE*. 523–534.

[60] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*. 1273–1282.

[61] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated Learning of Deep Networks using Model Averaging. *CoRR* abs/1602.05629 (2016).

[62] C. Mohan. 2019. State of Public and Private Blockchains: Myths and Reality. In *SIGMOD*. 404–411.

[63] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE Symposium on Security and Privacy*. 19–38.

[64] Piero Molino, Yaroslav Dudin, and Sai Sumanth Miryala. 2019. Ludwig: a type-based declarative deep learning toolbox. *CoRR* abs/1909.07930 (2019).

[65] Peter Müllner, Dominik Kowald, and Elisabeth Lex. 2021. Robustness of Meta Matrix Factorization Against Strict Privacy Constraints. *CoRR* abs/2101.06927 (2021).

[66] Milos Nikolic, Mohammed Elseidy, and Christoph Koch. 2014. LINVIEW: incremental view maintenance for complex analytical queries. In *SIGMOD*. 253–264.

[67] Randal S. Olson and Jason H. Moore. 2019. TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. In *Automated Machine Learning*. 151–160.

[68] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.

[69] Arnab Phani, Benjamin Rath, and Matthias Boehm. 2021. LIMA: Fine-grained Lineage Tracing and Reuse in Machine Learning Systems. In *SIGMOD*.

[70] Christopher Ré et al. 2020. Overton: A Data System for Monitoring and Improving Machine-Learned Products. In *CIDR*.

[71] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konecný, Sanjiv Kumar, and H. Brendan McMahan. 2020. Adaptive Federated Optimization. *CoRR* abs/2003.00295 (2020).

[72] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.

[73] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *SCIPY*.

[74] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. 2019. Clustered Federated Learning: Model-Agnostic Distributed Multi-Task Optimization under Privacy Constraints. *CoRR* abs/1910.01991 (2019).

[75] Sebastian Schelter. 2020. "Amnesia" - Machine Learning Models That Can Forget User Data Very Fast. In *CIDR*.

[76] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *NeurIPS*. 2503–2511.

[77] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing Data Science through Interactive Curation of ML Pipelines. In *SIGMOD*. 1171–1188.

[78] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. 2015. Incremental Knowledge Base Construction Using DeepDive. *PVLDB* 8, 11 (2015), 1310–1321.

[79] Alexander J. Smola and Shravan M. Narayanamurthy. 2010. An Architecture for Parallel Topic Models. *PVLDB* 3, 1 (2010), 703–710.

[80] Evan R. Sparks, Ameet Talwalkar, Daniel Haas, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. 2015. Automating model search for large scale machine learning. In *SoCC*. 368–380.

[81] Evan R. Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J. Franklin, and Benjamin Recht. 2017. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. In *ICDE*. 535–546.

[82] Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. 2018. D^2: Decentralized Training over Decentralized Data. In *ICML (Proceedings of Machine Learning Research)*, Vol. 80. 4855–4863.

[83] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *SIGKDD*. 847–855.

[84] Yuanyuan Tian, Fatma Özcan, Tao Zou, Romulo Goncalves, and Hamid Pirahesh. 2016. Building a Hybrid Warehouse: Efficient Joins between Data Stored in HDFS and Enterprise Warehouse. *ACM Trans. Database Syst.* 41, 4 (2016), 21:1–21:38.

[85] Tiffany Tuor, Shiqiang Wang, Bong Jun Ko, Changchang Liu, and Kin K. Leung. 2020. Overcoming Noisy and Irrelevant Data in Federated Learning. *CoRR* (2020).

[86] Stef van Buuren and Karin Groothuis-Oudshoorn. 2011. mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software, Articles* 45, 3 (2011), 1–67.

[87] Manasi Vartak, Joana M. F. da Trindade, Samuel Madden, and Matei Zaharia. 2018. MISTIQUE: A System to Store and Query Model Intermediates for Model Diagnosis. In *SIGMOD*. 1285–1300.

[88] Manasi Vartak and Samuel Madden. 2018. MODELDB: Opportunities and Challenges in Managing Machine Learning Models. *IEEE Data Eng. Bull.* 41, 4 (2018), 16–25.

[89] Ashish Vulimiri, Carlo Curino, Brighten Godfrey, Konstantinos Karanasos, and George Varghese. 2015. WANalytics: Analytics for a Geo-Distributed Data-Intensive World. In *CIDR*.

[90] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter R. Pietzuch. 2016. Ako: Decentralised Deep Learning with Partial Gradient Exchange. In *SoCC*. 84–97.

[91] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *SciPy*. 56–61.

[92] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy Preserving Vertical Federated Learning for Tree-based Models. *PVLDB* 13, 11 (2020), 2090–2103.

[93] Doris Xin, Litian Ma, Jialin Liu, Stephen Macke, Shuchen Song, and Aditya G. Parameswaran. 2018. Helix: Accelerating Human-in-the-loop Machine Learning. *PVLDB* 11, 12 (2018), 1958–1961.

[94] Doris Xin, Stephen Macke, Litian Ma, Jialin Liu, Shuchen Song, and Aditya G. Parameswaran. 2018. Helix: Holistic Optimization for Accelerating Iterative Machine Learning. *PVLDB* 12, 4 (2018), 446–460.

[95] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* 10, 2 (2019), 12:1–12:19.

[96] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.* 41, 4 (2018), 39–45.

[97] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI*. 15–28.

[98] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M. Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. 2020. The NebulaStream Platform for Data and Application Management in the Internet of Things. In *CIDR*.

[99] Ce Zhang, Arun Kumar, and Christopher Ré. 2014. Materialization Optimizations for Feature Selection Workloads. In *SIGMOD*. 265–276.

[100] Hantian Zhang, Luyuan Zeng, Wentao Wu, and Ce Zhang. 2017. How Good Are Machine Learning Clouds for Binary Classification with Good Features? *CoRR* abs/1707.09562 (2017).

[101] Jingren Zhou, Per-Åke Larson, and Hicham G. Elmongui. 2007. Lazy Maintenance of Materialized Views. In *VLDB*. 231–242.