Federated Data Preparation, Learning, and Debugging in Apache SystemDS

Sebastian Baunsgaard
Graz University of Technology
Graz, Austria

Matthias Boehm
Graz University of Technology
Know-Center GmbH
Graz, Austria

Kevin Innerebner
Graz University of Technology
Graz, Austria

Mito Kehayov
Florian Lackner
Olga Ovcharenko
Graz University of Technology
Graz, Austria

Arnab Phani
Tobias Rieger
David Weissteiner
Graz University of Technology
Graz, Austria

Sebastian Benjamin Wrede
Graz University of Technology
Know-Center GmbH
Graz, Austria

ABSTRACT

Federated learning allows training machine learning (ML) models without central consolidation of the raw data. Variants of such federated learning systems enable privacy-preserving ML, and address data ownership and/or sharing constraints. However, existing work mostly adopt data-parallel parameter-server architectures for mini-batch training, require manual construction of federated runtime plans, and largely ignore the broad variety of data preparation, ML algorithms, and model debugging. Over the last years, we extended Apache SystemDS by an additional federated runtime backend for federated linear-algebra programs, federated parameter servers, and federated data preparation. In this paper, we share the system-level compiler and runtime integration, new features such as multi-tenant federated learning, selected federated primitives, multi-key homomorphic encryption, and our monitoring infrastructure. Our demonstrator showcases how composite ML pipelines can be compiled into federated runtime plans with low overhead.

CCS CONCEPTS

• **Information systems → Data management systems**; • **Computing methodologies → Machine learning**.

KEYWORDS

Federated Learning, Federated Raw Data, Monitoring

ACM Reference Format:
Sebastian Baunsgaard, Matthias Boehm, Kevin Innerebner, Mito Kehayov, Florian Lackner, Olga Ovcharenko, Arnab Phani, Tobias Rieger, David Weissteiner, and Sebastian Benjamin Wrede. 2022. Federated Data Preparation, Learning, and Debugging in Apache SystemDS. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557162

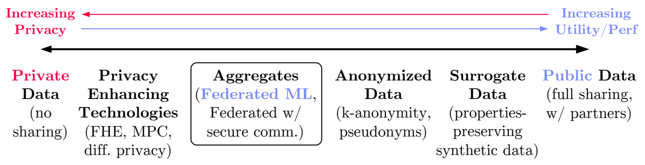

Increasing Privacy
Increasing Utility/Perf
Private Data (no sharing)
Privacy Enhancing Technologies (FHE, MPC, diff. privacy)
Aggregates (Federated ML, Federated w/ secure comm.)
Anonymized Data (k-anonymity, pseudonyms)
Surrogate Data (properties-preserving synthetic data)
Public Data (full sharing, w/ partners)

Figure 1: Spectrum of Data Sharing.

1 INTRODUCTION

Data privacy requirements, data ownership, and other constraints on data sharing render central data consolidation—and training machine learning (ML) models on this data–infeasible in many applications. For this reason, privacy-preserving ML and data sharing have been addressed with a variety of techniques. Figure 1 shows the resulting *spectrum of data sharing*, covering techniques with different tradeoffs of privacy, performance, and utility for ML applications. Private data and heavy-weight techniques like fully homomorphic encryption (FHE) [2], multi-party computation (MPC) [21], and differential privacy [15] provide the strongest privacy guarantees with limited utility. Other techniques like anonymized or surrogate data [9] are more prone to revealing the raw data.

Federated learning: In contrast, federated learning [7, 16] overcomes sharing limitations by training models on the decentralized, federated data. Often data-parallel parameter servers [11, 18, 27] and similar distribution strategies [1, 19] are adopted [16]. A central coordinator initializes the model; federated devices or sites now act as workers, pull the current model, perform, for instance, a forward and backward pass through a neural network to compute gradients, and push these gradients or updated models to the coordinator, where the updates are accumulated and shared with all workers. Federated learning adds an interesting and practical design point to the spectrum of data sharing by sharing aggregates—that reveal data distributions but not the raw data—with moderate overhead.

Limitations of Existing Work: Existing work on federated learning made valuable contributions to system infrastructure [7], optimization algorithms [16, 23, 25], and the integration with privacy enhancing technologies [12, 13, 20, 29, 31]. However, existing work primarily focuses on data-parallel parameter servers (for mini-batch training), which ignores data preparation and feature transformations, a wide variety of batch ML algorithms, as well as end-to-end ML pipelines. Systems like TensorFlow federated [14] allow for general federated analytics, but require users or developers to construct federated runtime plans. Creating new libraries of federated ML algorithms and primitives [12, 13, 29] becomes difficult for complex, composite ML pipelines.

Federated Learning in SystemDS: A key observation underpinning Apache SystemDS [4] is that state-of-the-art data cleaning and debugging techniques are based on machine learning, and thus, can be expressed in linear algebra. For this reason, we build a hierarchy of built-in functions for data preparation, training, and debugging on top of a domain-specific language for ML training and scoring. An optimizing compiler then generates efficient hybrid runtime plans of local, in-memory and distributed operations. This design allows a seamless integration of federated learning by dedicated compilation and runtime techniques. In the larger ExDRa project, SystemDS is already used as the backbone for federated learning [3]. Key differentiators are generic abstractions for federated data and request types; federated instructions for a variety of linear algebra operations and statistical functions; federated parameter servers; federated data preparation; as well as the automatic generation of valid and efficient federated runtime plans.

Contributions: In this demonstration proposal, we share the end-to-end system integration of SystemDS' federated backend, including new features on compiling federated plans, multi-tenant federated learning, and selected federated primitives (Section 2), as well as a monitoring tool for federated learning (Section 3). All features are fully integrated in Apache SystemDS[1]. The demonstration scenarios (Section 4) then utilize these components and provide an in-depth understanding of how ML pipelines are compiled into federated runtime plans, and how these plans are executed.

2 SYSTEM ARCHITECTURE

In this section, we recap the system architecture of Apache SystemDS' federated backend [3] and describe recent extensions of handling federated data, compiling federated runtime plans, multi-tenant federated learning, and selected federated operations.

2.1 Federated Runtime Backend

Federated Data: Our federated backend comprises abstractions for federated data, federated linear algebra operations for ML algorithms, federated parameter servers for mini-batch training, and federated feature transformations [3]. Federated frames or matrices are virtual objects whose physical parts are located at SystemDS workers in federated sites. Such federated matrices can be read from meta data, or constructed at script level:

```
F = federated(
      addresses=list("node1:8001/finput1.csv",
                     "node2:8001/finput2.csv"),
      ranges=list(list(0,0), list(50K,70),  #50K rows
             list(50K,0), list(120K,70)));  #70K rows
F = scale(X=F, center=TRUE, scale=TRUE);
[C,Y] = kmeans(X=F, k=4, runs=10, max_iter=20);
```

User scripts at the coordinator (e.g., scale and k-Means above) are compiled into a hierarchy of program blocks and instructions, and all instructions on federated data become federated operations.

[1]The source code is available at https://github.com/apache/systemds, in packages runtime/controlprogram/federated, runtime/instructions/fed, and hops/fedplanner.

Federated Requests: Federated operations are implemented via a small set of *federated requests* (READ_VAR, PUT_VAR, GET_VAR, EXEC_INST, EXEC_UDF, and CLEAR). For example, a matrix-vector multiplication on row-partitioned federated data broadcasts the vector via PUT_VAR, executes partial matrix-vector multiplications via EXEC_INST, and optionally obtains and concatenates the federated outputs. Batches of these requests are executed concurrently via Netty remote procedure calls (RPCs) for all federated workers.

Partitioning and Replication: Federated data allows for arbitrary disjoint partitioning. In order to simplify federated operations, we employ specific partitioning and replication types (FType). In this context, a row-partitioned federated matrix has an FType.ROW (partitioning ROW, replication NONE), while a broadcast variable has FType.BROADCAST (partitioning NONE, replication FULL).

Broadcasting: Broadcasting in a federated setting is similar to broadcasting in data-parallel frameworks like Spark [30] or Dask [24]. However, meta data about federated ranges allows more control. We provide internal primitives for broadcast and broadcastSliced. For a row-partitioned matrix, a matrix-vector multiplication needs the full vector at every worker, whereas a vector-matrix multiplication only needs vector slices, avoiding unnecessary data transfer and memory consumption. Furthermore, in contrast to RDD and broadcast variables in Spark, all broadcasts are managed as federated data. Thus, a sliced broadcast simply becomes a federated matrix, and all federated operations apply.

2.2 Compiling Federated Plans

Besides a basic runtime conversion—which robustly works even for conditional control flow—we now support configurable types for compiling federated runtime plans. The compilation of federated plans facilitates debugging, controls the execution of cost-optimal plans, and adheres to user-provided privacy constraints.

- *None:* Obtain the federated data and run local operations.
- *Runtime:* Convert all operations on federated inputs to federated operations during runtime (Section 2.1) [3].
- *Compile-Fed_All:* Compile federated operations with the objective of keeping intermediates federated if possible.
- *Compile-Fed_Heuristic:* Compile federated operations with the same heuristics as *Runtime* (e.g., collect agg. vectors).
- *Compile-Fed_Cost-based:* Recursively enumerate optimal subplans for interesting properties of intermediates (e.g., FType.ROW, FType.COL, Local) and finally, select the global cost-optimal plan from terminal operators (write and print).

Table 1 shows the runtime of L2-SVM (support vector machines) on federated data, where Fed-All is—even in a local area network—almost 2x slower than Fed-Heuristic/Fed-Cost due to the RPC latency of many federated vector operations for a line search on the gradients in an inner loop. Instead, these operations can be performed at the coordinator because the gradients are aggregates.

Enumeration Mechanics: After extracting the federated data specification from reads or federated initializations, we obtain the

Table 1: L2SVM Training w/ Federated Planners (3 workers).

Local	Fed-None	Fed-Runtime	Fed-All	Fed-Heuristic
5.2s (9.6s)	89.2s	5.9s	9.4s	5.8s

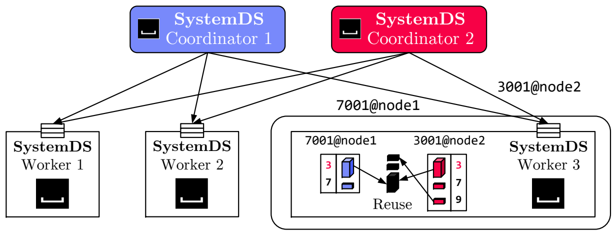


Figure 2: Multi-tenant Federated Learning with Reuse.

specific FType of inputs. The federated planners then iterate over the program structure, and propagate the type of intermediates. Operations can be forced to federated operations, and their outputs can be fixed to local or federated. Additional primitives exist for collecting (fed-to-local) and broadcasting (local-to-fed) data.

Privacy Constraints: We further support propagating and checking privacy constraints—both during compilation and runtime—from the inputs through the program. Example constraints include types such as public, private-aggregate, and private partitions or features; and valid purposes such as specific operations. If existing, these constraints change the objective to a constrained minimization of cost subject to these privacy constraints, which raises errors if no plan satisfies the constraints. In the future, we will also integrate privacy enhancing technologies (e.g., homomorphic encryption) in this framework to handle strictly private data.

2.3 Multi-tenant Federated Learning

Federated workers are long-running server processes that receive concurrent requests from multiple coordinators. This scenario requires robust multi-tenant federated learning as shown in Figure 2.

Tenant Isolation: For robust isolation, we use a tree of tenant-specific execution contexts, where each context also holds a map of life variables (mapping from variable ID to unpinned data objects). Tenants are uniquely identified by a combination of their coordinator process ID (included in requests) and IP address as seen from the worker. For each request, we construct the coordinator ID, lookup the tenant context, and read/store intermediates in this context. This approach guarantees that no intermediates are overwritten, even if coordinators run on the same host. Furthermore, a single coordinator might run a parallel for-loop and spawn multiple concurrent requests. The parfor worker IDs are also included in the requests and create an additional hierarchy level of contexts.

Event-Loop: Workers listen for federated requests, execute these requests, and return federated responses with optional SSL-encryption. The configuration of this event-loop has high impact on performance. By default, we limit the number of concurrent connections to the number of virtual cores because typically the number of coordinators is moderate. For incoming `EXEC_*` requests, the workers further set the operation parallelism to the number of virtual cores for adapting to heterogeneous worker hardware and avoiding under-utilization with time-varying workloads.

Lineage-based Reuse: Fully isolating the individual tenants leads to unnecessary redundancy. First, the federated data is read and kept in memory multiple times. Second, exploratory data science exhibits high redundancy in and across ML pipelines of a single tenant, but also across multiple tenants. We address this challenge by a dedicated read cache and lineage-based reuse [22] with placeholders for synchronization during execution. Files are assumed unchanged during uptime of federated workers. We then trace lineage of all operations, which takes the lineage of inputs and constructs the output lineage. This lineage is the key in a process-wise lineage cache across all contexts. For obtaining the lineage of received data (in `PUT_VAR` requests), coordinators can send the data's lineage, and if unavailable, we construct a high-probability identifier based on CRC checksums and additional meta data.

2.4 Selected Federated Primitives

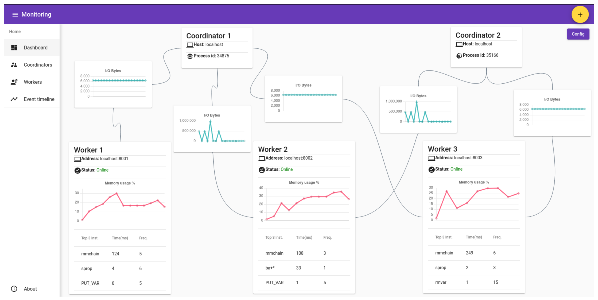
Basic Federated Operations: Meanwhile SystemDS reached feature completeness for all operations that can be supported on federated data. Basic operations that map cleanly to federated operations include feature transformations (encode, apply, decode), a variety of linear algebra and arithmetic operations, statistical functions, aggregations, indexing, and reorganizations. Recently, we also added fused operator pipelines [6], and second-order operations like `map(F, v -> v.substring(5))`. With these federated operations, many DSL-based built-in functions for data preparation, ML algorithms, and model debugging can process federated data.

Advanced Federated Operations: Several operations require multi-pass implementations. These operations include cumulative aggregates, and remove empty row/column operations, which have cross-row/-column dependencies but can be efficiently computed via aggregation trees [5]. Other operations like ordering are impossible on row-partitioned data. Quantiles—which in turn rely on ordering—are, however, the basis for many statistics, data cleaning primitives (e.g., outlier removal), and equi-height histograms. Accordingly, we support federated quantiles via a recursive histogram refinement approach. First, we determine the min and max values of a row-partitioned feature and split this range into 256 buckets. Second, every federated worker counts the frequency of values per bucket on its local data. Third, the coordinator collects and aggregates these histograms. On the cumulative sum of these bucket frequencies, we determine in which bucket the required quantile falls (e.g., 0.5 quantile for the median). This bucket is then again split into 256 buckets and we repeat steps 2 and 3 until we get the result. Due to the large fan-out, federated quantiles only require few round-trips and the transferred data is very small.

Federated Parameter Server: For mini-batch training, SystemDS provides a dedicated `paramserv` built-in function for data-parallel (multi-threaded or distributed) parameter-servers. This parameter server infrastructure has been extended for federated operations by respecting the boundaries of federated data, and handling data imbalance (different data size) and skew (different data distribution). With parameters for update frequencies, this infrastructure is able to run FedAvg and similar optimization algorithms [16, 23]. Recently, we added support for multi-key homomorphic encryption [20] of gradients, which relies on Microsoft's SEAL library [8, 26], and guards against plain-text exchange of gradients.

3 MONITORING INFRASTRUCTURE

As an additional debugging and demonstration tool, we are currently building a dedicated monitoring infrastructure for Apache SystemDS Federated. In this section, we describe the backend and frontend (see Figure 3) designs of this infrastructure in detail.



Coordinator 1
Host: localhost
Process id: 34875

Coordinator 2
Host: localhost
Process id: 35166

Worker 1
Address: localhost:8001
Status: Online

Top 3 Inst.	Time(ms)	Freq.
mmchain	124	5
sprop	4	6
PUT_VAR	0	5

Worker 2
Address: localhost:8002
Status: Online

Top 3 Inst.	Time(ms)	Freq.
mmchain	108	3
ba+*	33	1
PUT_VAR	1	5

Worker 3
Address: localhost:8003
Status: Online

Top 3 Inst.	Time(ms)	Freq.
mmchain	249	6
sprop	2	3
rmvar	1	15

Figure 3: SystemDS Federated – Monitoring Frontend.

3.1 Monitoring Services

Infrastructure Registration: Given the nature of long-running federated workers, the monitoring service is configured with a set of workers (via IP and port) to monitor. Monitoring interactions are performed through `EXEC_UDF` requests, and the workers have configurations for accepting and answering these requests. In contrast, coordinators may point to a running monitoring service and register and deregister themselves for monitoring.

Statistics Monitoring and Aggregation: Once the connections are established, the monitoring service sets up groups of statistics for tracking at the different locations. These groups include node-specific statistics such as live data objects, memory consumption, executed instructions (with frequency/times), JVM statistics (e.g., JIT compilation and GC frequency/times), as well as channel-specific statistics such as histograms of request types, and the number and size of incoming/outgoing data transfers. These statistics are periodically collected, aggregated, and optionally logged.

3.2 User Interface and Debugging

Infrastructure View: Figure 3 shows the infrastructure view canvas of registered coordinators and workers as well as their active communication channels. This canvas already provides an overview of memory consumption and data transfers over time, along with basic statistics such as the number of heavy hitter instructions at the federated workers. This canvas is also the entry point for more details via the node and channel views.

Node and Channel Views: The summary views for individual nodes (coordinators and workers) as well as individual or aggregate channels then show the detailed statistics mentioned in Section 3.1. This drill-down to individual nodes, channels, or channel groups (e.g., all channels of coordinator 2) then allows detailed debugging of potential performance problems, data transfer characteristics, and suspicious behavior. Additionally, these views also provide—inspired by Spark [30]—event timelines with different tenants in separate lanes, color-encoding for different federated request types, and the length of events indicating their runtime.

4 DEMONSTRATION SCENARIOS

This demonstration will utilize Apache SystemDS, its federated backend, and the monitoring tool as is, allowing the audience to clone the public repository and reproduce the scenarios as needed. In detail, we plan the following setup and scenarios.

Demo Setup: Our example scenario setup involves three (or more) federated workers, which are started via shell scripts either locally on laptops at the demonstration site or in a cluster in Graz, Austria for showcasing wide-area-network access. Theses workers have access to prepared real and synthetic datasets (row-partitioned across federated workers, but also other partitioning schemes):

- *Public Data:* We use the Adult (census data) [28], Nashville (traffic accidents) [10], and Criteo (click logs) [17] datasets as a basis for our demonstration of federated ML pipelines.
- *Paper Production:* In addition, we use an anonymized paper production (paper quality) dataset from our industry partner [3], representing federated data across production sites.

On local laptops, we then start the monitoring tool as well as one or multiple coordinators for simulating data scientists exploring the federated data and developing ML pipelines.

Single-tenant Scenario: A first scenario shows a single coordinator working alone on the federated data. First, we start with individual operations (e.g., `colSums(X)` or `quantile(X, 0.95)`) and basic DSL-based built-in functions (e.g., `outlierByIQR(X=X, k=1.5)`). With the help of SystemDS' explain functionality and the monitoring tool, the audience will be able to observe and understand federated runtime plans, federated operations, and their characteristics (including data transfers). The coordinator scripts can be changed in seconds, allowing the interactive exploration of workloads (including end-to-end ML algorithms for batch and mini-batch training) and their runtime and accuracy characteristics.

Multi-tenant Scenario: A second scenario then brings up multiple tenants that work concurrently with the federated data. These tenants run entire ML pipelines (some of which use the same preprocessing primitives), hyper-parameter optimization, and cross-validation. In these more complex and partially redundant scenarios, we will show how the reuse cache is populated, as well as how reuse and evictions happen over time. Furthermore, the event timelines and summary of data objects also enables a deeper understanding of tenant isolation. Finally, there are plenty of configurations (e.g., BLAS libraries, event-loop configs, SSL-encrypted communication) that can be explored based on the interests of the audience.

5 CONCLUSIONS

In this demonstration, we presented the—now feature-complete—federated backend of Apache SystemDS for compiling and executing federated end-to-end ML pipelines. New components include the compilation of federated runtime plans, multi-tenant federated learning with robust isolation and reuse, new federated primitives for data preparation and debugging, as well as a new monitoring infrastructure. In contrast to existing systems, federated learning in SystemDS seamlessly applies to a wide variety of DSL-based ML algorithms and other primitives, reuses existing runtime infrastructure, and thus makes federated learning practical.

ACKNOWLEDGEMENTS

This work is a result of the ExDRa project, funded through the bilateral program "ICT of the Future – Smart Data Economy" by the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK, 873838). Contributions by Sebastian B. Wrede were made in the DDAI COMET module (funded by BMK, BMDW, FFG, SFG, and industrial partners).

REFERENCES

[1] Martín Abadi et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*. 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[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. https://doi.org/10.1145/3214303

[3] Sebastian Baunsgaard et al. 2021. ExDRa: Exploratory Data Science on Federated Raw Data. In *SIGMOD*. 2450–2463. https://doi.org/10.1145/3448016.3457549

[4] Matthias Boehm et al. 2020. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR*. http://cidrdb.org/cidr2020/papers/p22-boehm-cidr20.pdf

[5] Matthias Boehm, Alexandre V. Evfimievski, and Berthold Reinwald. 2019. Efficient Data-Parallel Cumulative Aggregates for Large-Scale Machine Learning. In *BTW*. 267–286. https://doi.org/10.18420/btw2019-17

[6] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Prithviraj Sen, Alexandre V. Evfimievski, and Niketan Pansare. 2018. On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML. *PVLDB* 11, 12 (2018), 1755–1768. https://doi.org/10.14778/3229863.3229865

[7] Keith Bonawitz et al. 2019. Towards Federated Learning at Scale: System Design. In *MLSys*. https://proceedings.mlsys.org/book/271.pdf

[8] Hao Chen, Kim Laine, and Rachel Player. 2017. Simple Encrypted Arithmetic Library - SEAL v2.1. In *Financial Cryptography and Data Security (Lecture Notes in Computer Science, Vol. 10323)*. Springer, 3–18.

[9] Graham Cormode and Divesh Srivastava. 2009. Anonymized data: generation, models, usage. In *SIGMOD*. 1015–1018. https://doi.org/10.1145/1559845.1559968

[10] Data.Nashville.gov. 2020. Nashville Traffic Accidents Dataset. https://data.nashville.gov/Police/Traffic-Accidents/6v6w-hpcw

[11] 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. https://proceedings.neurips.cc/paper/2012/hash/6aca97005c68f1206823815f66102863-Abstract.html

[12] Fangcheng Fu, Yingxia Shao, Lele Yu, Jiawei Jiang, Huanran Xue, Yangyu Tao, and Bin Cui. 2021. VF^{2}Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning. In *SIGMOD*. 563–576. https://doi.org/10.1145/3448016.3457241

[13] Fangcheng Fu, Huanran Xue, Yong Cheng, Yangyu Tao, and Bin Cui. 2022. BlindFL: Vertical Federated Machine Learning without Peeking into Your Data. In *SIGMOD*. 1316–1330. https://doi.org/10.1145/3514221.3526127

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

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

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

[17] Criteo AI Lab. 2020. Criteo 1TB Click Logs Dataset. https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/

[18] 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. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[19] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *PVLDB* 13, 12 (2020), 3005–3018. https://doi.org/10.14778/3415478.3415530

[20] Jing Ma, Si-Ahmed Naas, Stephan Sigg, and Xixiang Lyu. 2022. Privacy-preserving federated learning based on multi-key homomorphic encryption. *Int. J. Intell. Syst.* 37, 9 (2022), 5880–5901. https://doi.org/10.1002/int.22818

[21] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE Symp. on Security and Privacy*. 19–38. https://doi.org/10.1109/SP.2017.12

[22] Arnab Phani, Benjamin Rath, and Matthias Boehm. 2021. LIMA: Fine-grained Lineage Tracing and Reuse in Machine Learning Systems. In *SIGMOD*. 1426–1439. https://doi.org/10.1145/3448016.3452788

[23] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. 2021. Adaptive Federated Optimization. In *ICLR*. https://openreview.net/forum?id=LkFG3lB13U5

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

[25] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. 2021. Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints. *IEEE Trans. Neural Networks Learn. Syst.* 32, 8 (2021), 3710–3722. https://doi.org/10.1109/TNNLS.2020.3015958

[26] SEAL 2022. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[27] Alexander J. Smola and Shravan M. Narayanamurthy. 2010. An Architecture for Parallel Topic Models. *PVLDB* 3, 1 (2010), 703–710. https://doi.org/10.14778/1920841.1920931

[28] UCI. 2020. Adult Data Set. https://archive.ics.uci.edu/ml/datasets/adult

[29] 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. https://doi.org/10.14778/3407790.3407811

[30] 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. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia

[31] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. 2020. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. In *ATC*. 493–506. https://www.usenix.org/conference/atc20/presentation/zhang-chengliang