Amalur
The Convergence of Data Integration and Machine Learning

Li, Ziyu; Sun, Wenbo; Zhan, Danning; Kang, Yan; Chen, Lydia; Bozzon, Alessandro; Hai, Rihan

DOI
10.1109/TKDE.2024.3357389

Publication date
2024

Document Version
Final published version

Published in
IEEE Transactions on Knowledge and Data Engineering

Amalur: The Convergence of Data Integration and Machine Learning

Ziyu Li, Wenbo Sun, Danning Zhan, Yan Kang, Lydia Chen, Alessandro Bozzon, and Rihan Hai

***Abstract*—Machine learning (ML) training data is often scattered across disparate collections of datasets, called *data silos*. This fragmentation poses a major challenge for data-intensive ML applications: integrating and transforming data residing in different sources demand a lot of manual work and computational resources. With data privacy constraints, data often cannot leave the premises of data silos; hence model training should proceed in a decentralized manner. In this work, we present a vision of bridging traditional data integration (DI) techniques with the requirements of modern machine learning systems. We explore the possibilities of utilizing metadata obtained from data integration processes for improving the effectiveness, efficiency, and privacy of ML models. Towards this direction, we analyze ML training and inference over data silos. Bringing data integration and machine learning together, we highlight new research opportunities from the aspects of systems, representations, factorized learning, and federated learning.**

***Index Terms*—Machine learning, data integration, federated learning.**

I. Introduction

THE accuracy of an ML model heavily depends on the training data. In real-world applications, often the data is not stored in a central database or file system but spread over different data silos. Examples include drug-risk prediction [1] or keyboard stroke prediction [2]; the data is collected at different locations or devices.

Data integration (DI) systems enable interoperability among multiple, heterogeneous sources and provide a unified view for users. Notably, they allow us to describe data sources and their relationships [3]: i) mappings between different source schemata, i.e., schema matching and mapping [4], [5] and ii) linkages between data instances, i.e., data matching (also known as record linkage or entity resolution (ER)) [6]. Nevertheless, a DI system aims to facilitate query answering or data transformation over silos, not directly supporting ML applications. As a result, practitioners nowadays tackle silos with DI systems and ML tooling separately.

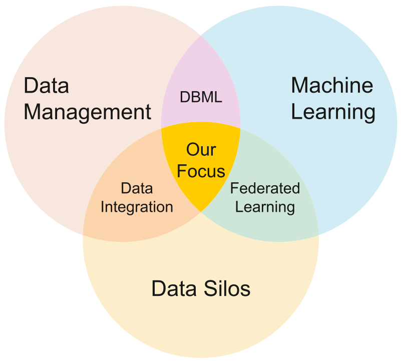


Fig. 1. Scope of this line of work.

Running Example: Consider the feature augmentation example in Fig. 2; the downstream ML task is to predict patients' mortality (binary classification) based on information scattered across tables maintained by different departments in the same hospital. Data from the ER department are stored in a base table $S_1(m,n,a,hr)$, which has the label column m $(mortality)$ and features a (age) and hr $(resting\ heart\ rate)$. To improve the model's accuracy, a data discovery system is employed to discover a related table $S_2(m,n,a,o,dd)$ (Fig. 2(b)), with information coming from the pulmonary department. This table brings information about a new feature column o $(oxygen)$, which shows the blood oxygen level. The label column and the selected feature columns constitute the schema of the table for downstream ML models, i.e., $T(m, a, hr, o)$, which we refer to as the *target table schema* or *mediated schema*.

Data Integration, Data Management and ML: Fig. 1 illustrates our problem scope. Recent advances with *in-database machine learning* [7], [8], [9], mainly consider a single database instead of data silos.[1] Traditional data integration, which

Manuscript received 31 August 2023; accepted 5 January 2024. Date of publication 23 January 2024; date of current version 13 November 2024. This publication is part of the project Understanding Implicit Dataset Relationships for Machine Learning (with project number VI.Veni.222.439) of the research programme NWO Talent Programme Veni which is (partly) financed by the Dutch Research Council (NWO). This work was supported by the European Union Horizon Programme call HORIZON-CL4-2022-DATA-01 under Grant 101093164 (ExtremeXP). Recommended for acceptance by Lei Chen. *(Ziyu Li and Wenbo Sun contributed equally to this work.) (Corresponding author: Ziyu Li.)*

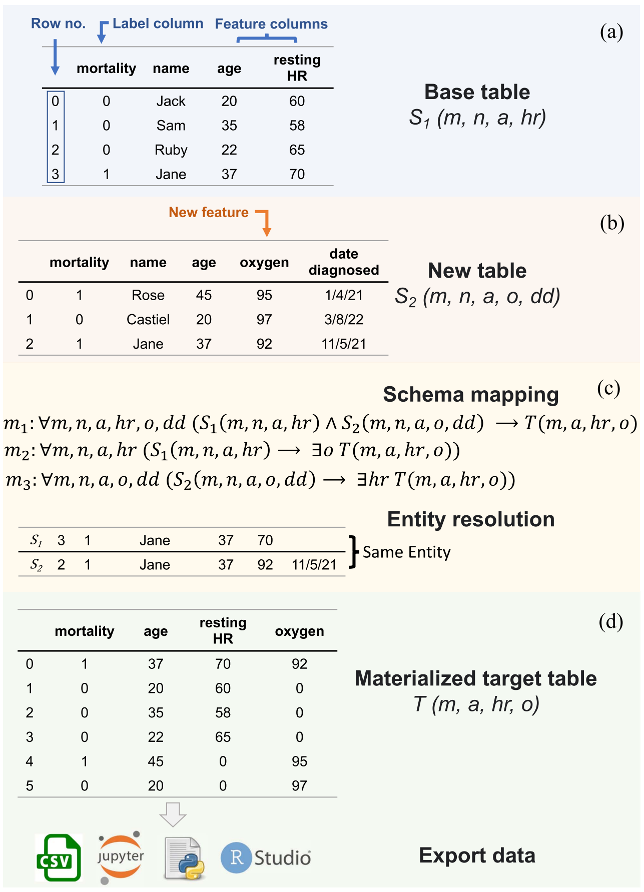
Ziyu Li, Wenbo Sun, Danning Zhan, Alessandro Bozzon, and Rihan Hai are with the Department of Software Technology, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: z.li-14@tudelft.nl; w.sun-2@tudelft.nl; d.zhan@tudelft.nl; a.bozzon@tudelft.nl; r.hai@tudelft.nl).

Yan Kang is with the WeBank, Shenzhen 518052, China (e-mail: yangkang@webank.com).

Lydia Chen is with the Department of Computer Science, University of Neuchatel, 2000 Neuchâtel, Switzerland, and also with the Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: L.chen-10@tudelft.nl).

Digital Object Identifier 10.1109/TKDE.2024.3357389

[1]The intersection of data management and ML (DBML) is two-fold: ML for DB, and DB for ML. ML has been applied to improve key operations of DI such as schema matching [10], [11], and data matching [11], [12], [13], [14]. In this paper, we focus on data management for ML. Except for data cleaning [15], [16], [17], little has been discussed in terms of using the key DI operations to facilitate ML [18].



	mortality	name	age	resting HR
0	0	Jack	20	60
1	0	Sam	35	58
2	0	Ruby	22	65
3	1	Jane	37	70

	mortality	name	age	oxygen	date diagnosed
0	1	Rose	45	95	1/4/21
1	0	Castiel	20	97	3/8/22
2	1	Jane	37	92	11/5/21

m_1: $\forall m, n, a, hr, o, dd\ (S_1(m,n,a,hr) \wedge S_2(m,n,a,o,dd)\ \longrightarrow T(m,a,hr,o)$

m_2: $\forall m, n, a, hr\ (S_1(m,n,a,hr) \longrightarrow\ \exists o\ T(m,a,hr,o))$

m_3: $\forall m, n, a, o, dd\ (S_2(m,n,a,o,dd) \longrightarrow\ \exists hr\ T(m,a,hr,o))$

S_1	3	1	Jane	37	70	
S_2	2	1	Jane	37	92	11/5/21

	mortality	age	resting HR	oxygen
0	1	37	70	92
1	0	20	60	0
2	0	35	58	0
3	0	22	65	0
4	1	45	0	95
5	0	20	0	97

Fig. 2. Traditional integration of data silos for ML.

addresses data management concerns across silos [19], do not inherently cater to the needs of ML. Federated learning (FL), on the other hand, involves training ML models using data residing in isolated silos [20], with an emphasis on data privacy regulations such as GDPR [21], HIPAA [22], IPA [23], and PIPL [24]. In this paper, we argue that when data management, data silos, and ML meet, there are new optimization opportunities and research challenges.

Issues With the Separation of Data Integration and ML: As shown in Fig. 2(c)–(d), to use the data from the two tables S_1 and S_2, a data scientist would need to rely on a DI system or manually find the schema mapping and entity resolution between the two given tables. We elaborate schema mappings in Section IV-A. Then, a DI system can integrate these source tables by merging the mapped columns and linked entities (i.e., matched rows). Finally, it materializes the data instances of the target table T and exports it to downstream ML applications. Such a process usually involves massive manual work and computation overhead, e.g., joining tables. The source datasets go through the long data integration and transformation pipeline and reach the ML training phase as one large materialized dataset. The rich structural and semantic metadata obtained during data integration is discarded.

Research Vision & Question: In this work, we explore the uncharted research area at the intersection of data integration and ML. We explore the possibilities of utilizing DI metadata (e.g., the output of *schema matching* and *entity resolution*) to improve ML model training and inference. As the starting point, we ask a fundamental question:

Q: Can we use DI metadata to improve the efficiency, effectiveness, and the privacy of ML model training and inference?

Data integration is a well-studied research area with mature logic-based theoretical frameworks, techniques, and systems [19], [25], [26]. Yet, current ML systems [27], [28] focus on ML pipelines, merging and materializing the source datasets being a common practice, using Python libraries such as Pandas, NumPy or SciPy. We envision novel systems that combine DI information to improve ML efficiency, effectiveness, and the privacy of data. We refer to efficiency as the overall training (inference) time, effectiveness as the accuracy of the ML models, and data privacy as not revealing new information to other parties. To this end, we present new research challenges to design such a novel DI-aware ML system. Moreover, we reveal exciting new research challenges when bridging data integration with ML philosophies, e.g., federated learning.

In this paper, we focus on these challenges in four aspects: *system design*, *metadata and their representations*, *computation efficiency*, and *privacy*. The contributions go as follows.

- *System design (Section II):* We present the design of *Amalur*, a novel ML system that leverages DI results. We elaborate on the research challenges of building such a system.
- *Heterogeneous metadata (Section III):* We showcase various types of metadata and highlight their roles in improving efficiency, effectiveness, and privacy for ML tasks.
- *DI metadata representations (Section IV):* We propose matrix-based representations for DI metadata, which capture i) column matches, ii) row matches, and iii) redundancies between data sources and the target table. We also discuss and compare other available alternatives as representations for DI metadata.
- *Computation efficiency: (Section V)* We highlight the new opportunities for utilizing DI metadata to improve the time-wise efficiency of ML model training over data silos.
- *Privacy (Section VI):* We discuss the research challenges of improving vertical federated learning with DI metadata.

II. Amalur: An DI-Aware ML System

In this section, we first explain the challenges of two common ML scenarios, i.e., feature augmentation and federated learning, and propose a novel system *Amalur* to tackle these challenges. Amalur is a unified ML system designed to enhance ML pipelines' efficiency, effectiveness, and privacy in data integration contexts. It facilitates model training and inference over data silos by leveraging DI metadata.

A. Silo Problem: DI Formulation for Different ML Scenarios

With two representative ML scenarios, we explain *when* training data could come from silos, i.e., feature augmentation and federated learning. As shown in Table I, for these scenarios, the dataset relationships between source tables and the desired target

TABLE I
FOUR EXAMPLE DATA INTEGRATION SCENARIOS FOR FEATURE AUGMENTATION AND FEDERATED LEARNING

No.	Dataset Relationship	Schema mappings	Example scenarios
1	Full outer join	$m_1:\ \forall m,n,a,hr,o,dd\ (S_1(m,n,a,hr) \wedge S_2(m,n,a,o,dd) \rightarrow T(m,a,hr,o))$ $m_2:\ \forall m,n,a,hr\ (S_1(m,n,a,hr) \rightarrow \exists o\ T(m,a,hr,o))$ $m_3:\ \forall m,n,a,o,dd\ (S_2(m,n,a,o,dd) \rightarrow \exists hr\ T(m,a,hr,o))$	Feature augmentation, Federated learning, ...
2	Inner join	$m_1:\ \forall m,n,a,hr,o,dd\ (S_1(m,n,a,hr) \wedge S_2(m,n,a,o,dd) \rightarrow T(m,a,hr,o))$	Feature augmentation, (Vertical) federated learning, ...
3	Left join	$m_1:\ \forall m,n,a,hr,o,dd\ (S_1(m,n,a,hr) \wedge S_2(n,a,o,dd) \rightarrow T(m,a,hr,o))$ $m_2:\ \forall m,n,a,hr\ (S_1(m,n,a,hr) \rightarrow \exists o\ T(m,a,hr,o))$	Feature augmentation, (Vertical) federated learning, ...
4	Union	$m_2:\ \forall m,n,a,hr,o\ (S_1(m,n,a,hr,o) \rightarrow T(m,a,hr,o))$ $m_3:\ \forall m,n,a,hr,o,dd\ (S_2(m,n,a,hr,o,dd) \rightarrow T(m,a,hr,o))$	Data sample augmentation, (Horizontal) federated learning, ...

table can be captured by a class of well-studied data dependencies, i.e., tuple-generating dependencies (tgds) [29], [30], which are the commonly used formalisms in data integration studies.

Scenario 1: Feature augmentation is the exploratory process of finding new datasets and selecting features that help improve the ML model performance [31], [32], [33]. Fig. 2(b) shows an example: starting from a base table S_1, we augment the features by introducing the table S_2 and selecting the new feature $o\ (oxygen)$.

Scenario 2: Federated learning (FL) [20] studies how to build joint ML models over data silos (e.g., enterprise data warehouses, edge devices) without compromising privacy, which follows a decentralized learning paradigm. Similar to the virtual data integration problem setting [3], FL assumes that source data is not stored at a central data store but stays locally. According to how the feature space and sample space are partitioned among the data sources, FL can be categorized as vertical federated learning (VFL) and horizontal federated learning (HFL) [34]. For VFL, data sources share the overlapping data instances, but the feature columns partially overlap. For HFL, data sources share the overlapping feature columns, while the data instances may overlap.

Example 1 (full outer join) is explained for feature augmentation in Fig. 2 and Example 4.1. Full outer join is also a general case of FL, where sources have similar schemata and data instances (entities) that may or may not overlap.

Example 2 (inner join) represents the data integration scenario where only overlapped rows in two sources will be transformed, i.e., an inner join between S_1 and S_2 followed by a projection on columns m, a, hr, o. It can be used to describe the feature augmentation processes where fewer missing values are preferred. Such dataset relationships also reside in a VFL context, where data sources share the sample space (overlapped rows) but not necessarily the feature space (overlapped feature columns).

Example 3 (left join) shows a left join between S_1 and S_2. Compared to Example 1, we slightly change the schema of S_2 by dropping the label column m. Example 3 describes another typical feature augmentation scenario for supervised learning: only the base table S_1 contains the label column. In VFL cases, not all but specific sources hold the labels for supervised model training.

Example 4 (union) is a special case of Example 1, where S_1 and S_2 do not share any rows. We modify the S_1 and S_2 schemata to share the same feature columns mapped to the target schema T. Example 4 can represent the scenario when a new table is selected to bring more data samples.

Two Computation Strategies: The training process of an ML model requires complex arithmetic computations. The computations in the previous four examples, i.e., joinable or unionable source tables, can be conducted in a *materialized* or *factorized manner*, similar to *data warehousing* and *virtual data integration*. Materialization requires joining the source tables and obtaining the instances of the target table before exporting it for model training, as depicted in Fig. 2. Another option is *learning over factorized joins* [35], also known as *factorized learning* [36]. Given an ML model and joinable tables of a database, factorized learning requires reformulating the ML model and pushing down the computation to each table. Compared to materialization, factorized learning does not affect model training accuracy but often helps to improve the training efficiency, as the arithmetic computation results after factorization, are the same as the original operators [35], [37], [38]. Similar to traditional DI systems, materialization is sometimes impossible due to privacy constraints and other reasons, which we address in Section VI. In this section, we focus on the performance implications of these two strategies.

B. Amalur Overview

We are currently developing *Amalur*, a *machine learning* system that is based on our work on data lakes [39] and model zoos [40]. With DI metadata, Amalur solves the challenges of efficient training and inference of ML models over data silos and reducing the manual work of integrating the data. Fig. 3 provides a high-level overview of Amalur with key components relevant to this paper. Our proposed system is designed to support both materialized and factorized data; however, while learning on materialized data is well established, our primary focus is to explore factorization.

User Inputs: Amalur empowers users, including domain experts like physicians or data scientists, to run predictive or ML model training tasks on data silos. Through the metadata provided by the catalog, users can choose the desired features and relevant data silos. They can also initiate model training using either custom models or Amalur's built-in ML models with metadata from the catalog. Furthermore, specific constraints, such as data privacy regulations like the GDPR [21], can be specified by individual users or particular silos.

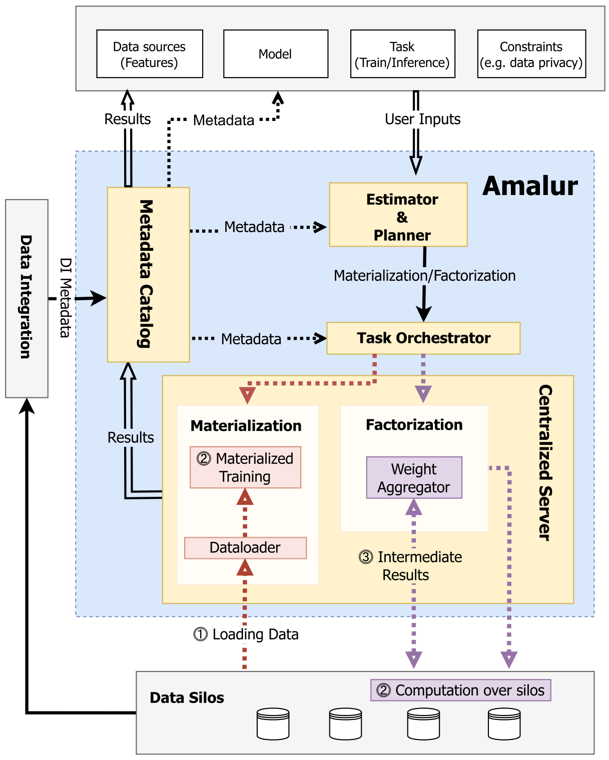


Fig. 3. Overview of Amalur.

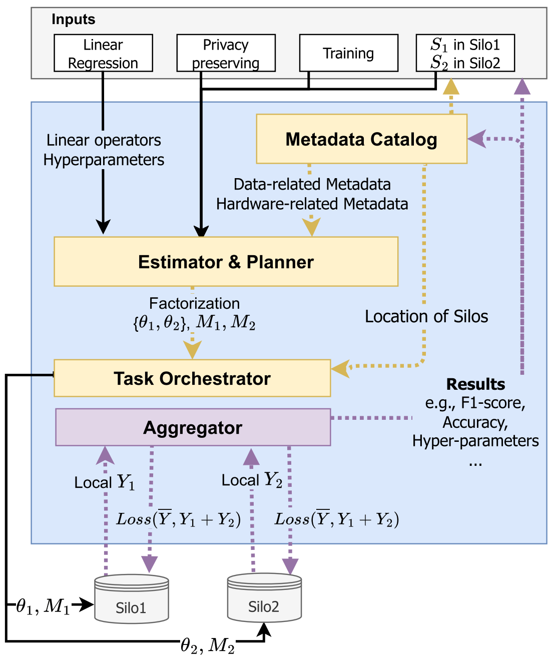


Fig. 4. Example workflows of factorized learning in Amalur.

Hybrid Metadata Catalog: One of the fundamental components of Amalur is the metadata catalog. It stores the metadata of data, ML models, and hardware settings. Data-related metadata includes the basic metadata and DI metadata. Collected from the silos, the basic metadata describes each data source, e.g., source table schema, data types, and silo location. The DI metadata includes column relationships from schema matching and row matching from entity resolution. Model-related metadata includes information describing models and the evaluation performance (e.g., model accuracy). We have addressed the representations of basic metadata of source tables [41] and ML models [40]. In this work, we focus on a mixture of heterogeneous metadata in Section III, and DI metadata with an explanation of their matrix-based representations in Section IV.

Estimator and Planner: The cost estimator and task planner play a critical role in the system. The cost estimator leverages the metadata from the catalog (e.g., data characteristics, hardware specifics) and constraints to determine the approach to execute the input task over silos: materialization and factorization. We have developed an initial cost estimator utilizing basic hardware information and the computational complexity of the target model. We elaborate on the cost estimator's theoretical foundations and preliminary results in Section V. The planner identifies the appropriate physical operators and generates an execution plan for task orchestration.

Task Orchestrator and Dispatcher: The execution plan from the planner is translated into specific programs tailored to the training approach, i.e., factorization, materialization, or FL, and the execution environment, such as TensorFlow, PyTorch, Spark, or ONNX. For materialization, *Dataloader* pulls data from the silos for processing. Model training or inference will be performed in the centralized server. Alternatively, if factorization is preferred, programs are sent to remote silos as the metadata dictates, i.e., silo location, ensuring they reach the appropriate data location. The main computations are performed over each silo.

Aggregator: For factorized learning and federated learning, a crucial component is the aggregator. Some computations are pushed to the silos while a central server aggregates the results. The computations are performed locally, and the parameters are learned globally. The role of the aggregator is to collect the result of local computations and then distribute the loss to the silos and aggregate the gradient of the parameters.

C. Amalur Workflows for ML Training and Inference

With the core components in Amalur being introduced, we will explain the main workflow among the components in Fig. 4. Given the user inputs in Amalur, different workflows are performed: either performing inference or training, either factorizing or materializing the data, etc.

Amalur allows users to determine the data sources and models. If there is available DI metadata, Amalur will provide data sources that can be connected. To increase the effectiveness of ML training, a user can select feature columns from the available schemata. Model-related metadata is also retrieved from the metadata catalog and provided to users, which enables them to decide what algorithm and hyper-parameters to use. Users may use their customized model and hyper-parameter sets. In the input phase, the user selects data sources (e.g., name of the table, name of the schema), chooses the model, and determines the task (classification or regression) and constraints (e.g., privacy).

With the inputs, model training or inference will be performed. In the end, all results, which include predictive outcomes and trained models, are gathered in a centralized cluster. Concurrently, the system logs the training or inference method (materialization/factorization), the hyper-parameters, and performance (e.g., F1-score, runtime) in the metadata catalog, making them accessible for future reference and used by other users. Below, we will introduce the training and inference in more detail.

Model Training: After Amalur receives the inputs from a user, the cost estimator will determine the computation strategy for training, i.e., to materialize or factorize, with metadata from the inputs and metadata catalog. For materialization, Amalur will integrate the source datasets and generate the target table in the centralized server, and training will be performed on the server. For factorization, the model is decomposed and pushed down to silos. When privacy constraints are present, Amalur executes privacy-aware model training processes over the silos [42], i.e., federated learning, which we elaborate in Section VI-A.

Fig. 4 depicts a workflow for ML model training in a factorized manner. The planner will split the model into the parameters θ_1, θ_2 along with the DI metadata M_1, M_2, which are pushed to Silo1 and Silo2 for computations respectively. Subsequently, the central server will collect the computations and aggregate the result, i.e., Y_1 from Silo1, computes the loss calculated from $Loss(\bar{Y}, Y_1 + Y_2)$ which is sent back to the silos for gradient updates. Once the loss meets a predefined criterion, a central orchestrator records performance metrics in the metadata catalog. In addition to illustrated workflow, due to privacy considerations in FL, the partial parameters are stored locally within the silos.

Model Inference: A user can select a specific model and perform model inference if models are available for the prepared dataset. Like model training, the cost estimator determines whether the computation is performed in a factorized or materialized manner. Model inference in a materialized manner is similar to model training. Inference in a factorized manner is slightly different, with only the local predictive results being sent to the centralized aggregator to generate the predictive results, while nothing is returned from the server.

III. Metadata in Amalur

Metadata is crucial for DI systems [3], [43], [44]. At the core of this vision, we reveal the importance of metadata, particularly DI metadata, for ML training and inference. In the following, we divide the relevant metadata into three categories, i.e., data-related, ML-related, and hardware-related metadata. In each category, we showcase the representative types of metadata in Table II, and discuss their roles in improving the effectiveness, efficiency, and privacy of ML model training and inference. The metadata is stored and managed by the metadata catalog described in Section II-B.

A. Data-Related Metadata

Metadata in databases and data lakes refers to the information that describes the structure, and characteristics of databases or data lakes and their objects [45], [46], [47]. The metadata includes information regarding schemata, statistical and descriptive data about relations and attributes, integrity constraints, silo locations, etc.

TABLE II
Example Metadata Types Used in Amalur

Category	Function	Metadata Type
Data-related Metadata	Dataset	Source location
		Schema (attribute, data type)
		Cardinality
	Data Integration	Schema mapping
		Row matching
Model-related Metadata	Model	Name
		Algorithm
		Hyper-parameters
	Trained Results	Training method (factorized/materialized)
		Parameters
		Performance (e.g., accuracy)
Hardware-related Metadata	Hardware and Environment	#GPU cores
		#CPU cores, cache sizes
		Memory bandwidth
		Memory latency

Data Integration Metadata: By DI metadata, we refer to the information that describes the relevance and overlap between data sources, e.g., schema-level correspondences between source tables and the target table (schema mapping), and row matching between source tables (entity resolution).[2]

To address the research question in Section I , we employ DI metadata in a threefold manner.

1. Efficiency: By representing schema mapping and row matching as matrices (Section IV), Amalur facilitates a unified execution of data transformation operations and linear algebra operations (Section V-A and VI-A), which improves the efficiency of training tasks (Sections V-B and V-C).

2. Effectiveness: DI metadata brings new opportunities for improving the effectiveness of federated learning frameworks, e.g., through discovering the redundancy among source datasets (Section VI-C).

3. Privacy: DI metadata leads to new challenges for data privacy in FL frameworks (Sections VI-B and VI-C).

Challenges: Another type of DI metadata is the similarity among source datasets. In recent studies on data lakes, it is a crucial step to first capture the similarity between source datasets, i.e., joinable or unionable dataset discovery [49], [50], [51], before data integration. The similarity between datasets is also valuable for improving the effectiveness of ML training, e.g., resolving the inconsistency across datasets and recommending models to train if the model was trained on a similar dataset [52]. In recent data integration works [11], [50], [53], the embeddings are applied to capture the similarities between features or tuples in source tables. Taking one step further, representations of the entire table [54], [55] allow for many more applications, e.g., transfer learning and multimodal machine learning. Many research questions remain open, regarding more types of DI

[2]How to obtain DI metadata is not the focus of this work, as schema matching and mapping, and entity resolution are intensively studied topics with open-source tools [12], [48] and commercial products. We assume that the DI metadata is part of Amalur's input. We are interested in how to utilize DI metadata for machine learning.

metadata, their representations, and their roles in improving ML tasks.

B. ML-Related Metadata

Amalur not only supports efficient ML training but also manages the trained models, which makes it necessary to design a metadata catalog that includes heterogeneous metadata for ML, i.e., ML-related metadata. ML-related metadata captures the metadata from various ML lifecycle stages [40], such as model definition and model training. The metadata describing the model includes architecture, framework, configurations (e.g., hyper-parameters), input/output (e.g., prediction classes), etc. These types of metadata are utilized in different components in Amalur. For example, the cost estimator requires information regarding the complexity of the model (e.g., algorithm, parameters) to predict workload and model training requires hyper-parameters. The metadata catalog also keeps track of the connections between the model and its training datasets. Thus, given a generated dataset, if a model was previously trained on this dataset, the model will be recommended to the user for inference or retraining.

Besides the information describing a model, we also record the performance of ML models (e.g., model accuracy, runtime, memory footprint) under different execution environments. This information helps recommend models to users in the model preparation stage. Recommending a good model to train based on different strategies [56] can improve the effectiveness and efficiency of ML training.

C. Metadata Regarding Hardware

Hardware and environment settings are important to measure the performance of databases [57], the interested information including, e.g., number of CPU cores, memory. This information is also essential for measuring ML performance during training and inference [58]. For ML, the hardware devices may also include GPU or TPU. In general, the hardware-related metadata is regarding the execution environment in the central cluster and distributed silos.

In this paper, the hardware-related metadata also supports an essential component for improving the efficiency of ML training. This type of metadata, along with the data-related and ML-related metadata, supports the cost estimator to decide whether to train a model on materialized or factorized data. The result of the cost estimation results in a more efficient plan for model training or inference.

Summary & Opportunities

– Metadata can be heterogeneous and serves different purposes.
– DI metadata provides opportunities to improve the efficiency, effectiveness and privacy for ML tasks.

IV. Representation: A Tale of Three Matrices

In this section, we discuss the representations that capture the metadata of DI, which enable optimizing ML tasks. When combining the functions of DI and ML in one system such as Amalur, one core task is how to represent the DI metadata. The challenge is that *the representation needs to be expressive enough to capture the DI metadata while bringing little overhead for model training*.

We define three logical-level representations: *mapping matrix* for preserving the column mapping (Section IV-A), *indicator matrix* for row matching (Section IV-B), and *redundancy matrix* for data redundancy (Section IV-C). We choose matrix-based representations, as they facilitate direct computation with linear algebras without the need of additional transformation, which we illustrate in Section V. Finally, in Section IV-D, we inspect the implementation of such matrix-based representations at the physical level.

A. Mapping Matrix

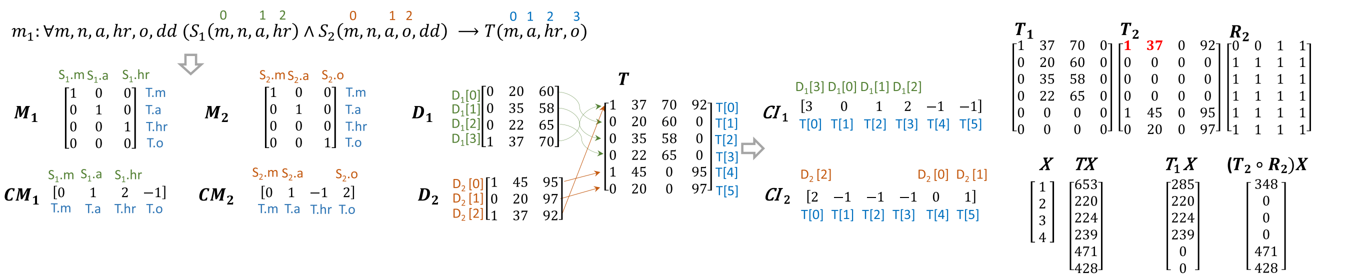
Preliminaries: Schema mappings lay at the heart of data integration and data exchange. Let $\mathbf{S}$ and $\mathbf{T}$ be a source relational schema and a target relational schema sharing no relation symbols. A *schema mapping* $\mathcal{M}$ between $\mathbf{S}$ and $\mathbf{T}$ is a triple $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$, where Σ is a set of dependencies over $\langle \mathbf{S}, \mathbf{T} \rangle$. The dependencies Σ can be expressed as logical formulas over source and target schemas. One of the most commonly used mapping languages is *source-to-target tuple generating dependencies (s-t tgd)* [29], [30], which are also known as Global-Local-as-View (GLAV) assertions [25]. An s-t tgd is a first-order sentence in the form of $\forall \mathbf{x}\ (\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y}))$ where $\varphi(\mathbf{x})$ is a conjunction of atomic formulas over the source schema $\mathbf{S}$,and, and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over the target schema $\mathbf{T}$.

Example 4.1: In Fig. 2(c), m_1, m_2, and m_3 are all tgds. We represent mapped attributes with the same variable names, e.g., $S_1.m$ and $S_2.m$. The tgd m_1 specifies that the overlapped rows of S_1 and S_2 are added to T ($\wedge$ denotes a natural join between S_1 and S_2); m_2 and m_3 indicate that all rows of S_1 and S_2 will be transformed to generate new tuples in T, respectively. Among the three tgds, it is the union relationship. The three tgds together, describe that the instances in T are obtained via a full outer join between the datasets S_1 and S_2.

Gaps: Tgds are first-order sentences specifying schema mappings. A DI system often generates schema mappings as executable data transformation scripts, e.g., SQL, which transform the source data instances and materialize the target table T. In contrast, the fundamental language of ML models is LA. To embed schema mappings in an end-to-end ML pipeline, we need a novel representation for schema mappings, which is compatible with algebraic computation in ML model training.

Matrix-Based Representation for Schema Mappings: Schema mappings contain the information about the mapped columns between source and target tables. We define the *mapping matrix* to preserve such column mappings. As a preparation step, we add ID numbers to mapped columns as shown in Fig. 5(a). In Table III, we summarize the notations used.

Definition 4.1 (Mapping Matrix): Mapping matrices between source tables $S_1, S_2, \ldots, S_n$ and target table T are a set of binary matrices $\mathbf{M} = \{M_1, \ldots, M_n\}$. M_k ($k \in [1, n]$) is a matrix with


m_1: ∀m, n, a, hr, o, dd (S_1(m, n, a, hr) ∧ S_2(m, n, a, o, dd) ⟶ T(m, a, hr, o)
M_1
M_2
CM_1
CM_2
D_1
D_2
T
CI_1
CI_2
T_1
T_2
R_2
X
TX
T_1X
$(T_2 \circ R_2)X$

(a) Mapping matrix generation (b) Compressed indicator matrix generation (c) Redundancy matrix and LMM rewrite

Fig. 5. Mapping, indicator, and redundancy matrices of the running example.

TABLE III
NOTATIONS USED IN THE PAPER

Notation	Description
T/S_k	Target table/the k^{th} source table
D_k	Processed k^{th} source table in matrix form
c_T/c_{S_k}	Number of mapped columns in T/S_k
r_T/r_{S_k}	Number of mapped rows in T/S_k
M_k/CM_k	Full/compressed mapping matrix for S_k
I_k/CI_k	Full/compressed indicator matrix for S_k
R_k	Redundancy matrix for S_k

the shape $c_T \times c_{S_k}$, where

$$M_k[i,j] = \begin{cases} 1, & \textit{if } j^{th} \textit{ column of } S_k \textit{ is mapped to} \\ & \textit{the } i^{th} \textit{ column of } T \\ 0, & \textit{otherwise} \end{cases}$$

Intuitively, in $M_k[i,j]$ the vertical coordinate i represents the target table column while the horizontal coordinate j represents the mapped source table column. A value of 1 in M_k specifies the existence of column correspondences between S_k and T, while the value 0 shows that the current target table attribute has no corresponding column in S_k. Fig. 5(a) shows the mapping matrices M_1 for S_1, and M_2 for S_2 of the running example.

It is easy to see that the binary mapping matrices are often sparse. To solve the sparsity problem we apply a more compressed form of mapping matrices as follows.

Definition 4.2 (Compressed Mapping Matrix): Compressed mapping matrices between source tables $S_1, S_2, \ldots, S_n$ and target table T are a set of row vectors $\mathbf{CM} = \{CM_1, \ldots, CM_n\}$. CM_k $(k \in [1,n])$ is a row vector of size c_T, where

$$CM_k[i] = \begin{cases} j, & \textit{if } j^{th} \textit{ column of } S_k \textit{ is mapped to} \\ & \textit{the } i^{th} \textit{ column of } T \\ -1, & \textit{otherwise} \end{cases}$$

We continue with the running example. Fig. 5(a) illustrates the compressed mapping matrices $\mathbf{CM} = \{CM_1, CM_2\}$. They can be directly generated from schema mappings without the generation of the mapping matrices $\mathbf{M} = \{M_1, M_2\}$.

B. Indicator Matrix

We use the *indicator matrix* [37] (denoted as I_k) to preserve the row matching between each source table S_k and the target table T. Similar to the mapping matrix, a binary indicator matrix could be very sparse, and its compressed form is preferred. Due to space restriction, we directly define the *compressed indicator matrix*.

Definition 4.3 (Compressed Indicator Matrix): Compressed indicator matrices between source tables $S_1, S_2, \ldots, S_n$ and target table T are a set of row vectors $\mathbf{CI} = \{CI_1, \ldots, CI_n\}$. CI_k $(k \in [1,n])$ is a row vector of size r_T, where

$$CI_k[i] = \begin{cases} j, & \textit{if the } j^{th} \textit{ row of } S_k \textit{ is mapped to} \\ & \textit{the } i^{th} \textit{ row of } T \\ -1, & \textit{otherwise} \end{cases}$$

Fig. 5(b) shows the row matching of the running example and the compressed indicator matrices, CI_1 and CI_2.

C. Redundancy Matrix

DI systems often need to handle data redundancy when multiple sources have overlapping values. Consider the example in Fig. 2, when a user queries how many patients aged above 30 are in S_1 and S_2, the correct answer is three instead of four. That is the overlapped row of *Jane* should be counted only once. Such redundancy resides in the projection of shared rows on the overlapped columns. Similarly, to support ML models we also need to detect redundancy to avoid repeated computation, which might lead to erroneous results. Thus, we propose a declarative representation to capture redundancy, i.e., *redundancy matrix*. To prepare for its definition, we first discuss how each source table contributes to the target table materialization. With the mapping matrix M_k and indicator matrix I_k, we can transform a source table D_k to an intermediate matrix with the same shape as T, denoted as T_k, $T_k = I_k D_k M_k^T$

Fig. 5(c) shows T_1 and T_2 of the running example. The red values in T_2 are the repeated values that already appeared in T_1. It is easy to see that T_k indicates the contribution from each source S_k. However, due to the aforementioned redundancy issue (red values in T_2), we cannot make a simple matrix addition to obtain the target table. For instance, $T_1 + T_2 \neq T$ in Fig. 5(b)–(c). This is why we need the redundancy matrix, which is defined below.

Definition 4.4 (Redundancy Matrix): A redundancy matrix R_k of source table S_k is a binary matrix with the shape of $r_T \times c_T$, where

$$R_k[i,j] = \begin{cases} 0, & \textit{if } T_k[i,j] \textit{ is redundant} \\ 1, & \textit{otherwise} \end{cases}$$

Note that before we say the data of a source table is redundant, we first need to specify which source table is the base table. For instance, in Example 1–3 of Table I, if we specify S_1 as the base table, then we consider the overlapped values in S_2 are redundant, and only need to generate a redundancy matrix for S_2. For completeness, we can consider that the redundancy matrix for the base table is an *all-ones matrix*, with all the element values equal to one. Fig. 5(c) shows R_2 for S_2, which is computed based on mapping and indicator matrices of S_1 and S_2.

D. Metadata Representation as Tensors

The three proposed types of matrices offer a novel perspective on the data processing pipeline, where we can describe data, and DI processes with LA. Aside from the intuitive 2D matrix representation, we can use *high-dimensional tensors* to integrate data and metadata. For example, the data matrix D_k can be expanded to a third dimension where M_k and I_k adhere along values and primary keys respectively. As such, we can represent the data and DI metadata with a more expressive data structure compatible with tensor algebra and recent advances in data processing [59], [60], [61], [62], [63], [64], [65].

As tensor processing modules become more prevalent, tensor representation aligns well with new hardware advancements. Both Google TPUs and recent Nvidia GPUs have built-in tensor cores, remarkably boosting linear algebraic computations such as matrix multiplications. In our recent work on empirical performance comparison [66], we have extensively analyzed the performance differences between classic hash join and matrix multiplication join (MMJoin), using the DI metadata discussed in this section. Leveraging the vast parallelism and dedicated tensor cores, MMjoin on GPUs outperforms classic GPU hash joins (by percentage), even with the inherently higher computational complexity of matrix multiplications. This underscores the potential of tensorized DI metadata in real-world DI tasks.

Moreover, as the fundamental language of ML, tensor algebra and the benefits it brings in efficiency [59], [60], [61] have attracted the considerable attention of the ML and DB research communities. Many recent works [62], [63], [64], [65] have started considering the integration of data processing and ML pipeline in unified tensor runtimes. Such a combination enables cross-optimizations between data processing and ML pipelines, a vital part of the Amalur system.

Summary & Opportunities

– DI metadata can be preserved in matrix representation.

– Representing DI metadata as tensors offers cross-optimization opportunities between DI and ML, and can help us leverage emerging tensor processing methods and hardware for speedup.

V. Algebraic Computation Over Silos

This section explores the research opportunities of model training across disparate data silos utilizing DI metadata. Early factorization works [35], [36], [67] required *manual* design for factorizing specific ML algorithms such as linear and logistic regression, and decision trees. Recent advances [37], [38], [68] facilitate more automation and optimization opportunities by decomposing ML models to basic building blocks of linear algebra (LA) operators, i.e., developing rewriting rules for LA operators and then pushing them down to joinable tables. Existing solutions for factorization over joins [37], [69], [70] mainly tackle inner joins. Our main contribution is to expand the dataset relationships to more integration scenarios, with left joins, full outer joins, and unions, cf. Table I.

Below, we first discuss the methods and challenges of generalizing existing factorization techniques from singular databases to multiple data sources. Second, we introduce an ML-based cost estimator, which leverages factorization and materialization and improves model training efficiency over silos. We also cover the existing cost estimation challenges, from the logical level to the hardware level.

A. Computation Challenge: DI Metadata for Factorization

As discussed in the previous section, factorized learning does not affect the model training accuracy but often helps to improve the efficiency compared to materialization. In the following, we explain how the DI metadata can generalize existing factorization techniques over silos, and discuss the new challenges it brings. To simplify the discussion, here we use the example of LA operator *left matrix multiplication (LMM)* and its rewrite rule from [37].

New Algebraic Rewriting Rules: Given a matrix θ with the size $c_T \times c_\theta$, the LMM of T and θ is denoted as $T\theta$. For better understanding, we use mapping/indicator matrices M_k/I_k below, although we generate and utilize their compressed forms CM_k and CI_k in practice. Equations below present an example of transforming an existing LMM rewriting [37] with our proposed rule.

$$T\theta \to I_1(D_1\theta[1:c_{S_1},]) + I_2(D_2\theta[c_{S_1}+1:c_T,])[37] \quad (1)$$

$$T\theta \to I_1 D_1 M_1^T\theta + ((I_2 D_2 M_2^T)\circ R_2)\theta \text{ [Amalur]} \quad (2)$$

We first compute $I_k D_k M_k^T$ for each source table. In this step, we reorder the matrix multiplication sequence to reduce computation overhead, similar to the join-order optimization in databases. The second step is detecting and removing duplicated computations through the redundancy matrices. For instance, we continue with the running example. Consider D_1 as the base table while D_2 is redundant. To obtain the correct final LMM result, here we can perform a Hadamard Product (element-wise multiplication) between $I_2 D_2 M_2^T$ and the redundancy matrix R_2. This way, we drop the redundant intermediate results indicated by the redundancy matrix R_2. Fig. 5(c) shows the results of $T_1\theta$ and $(T_2 \circ R_2)\theta$. Verifying that their addition is the same as $T\theta$ is easy.

DI Metadata & Operator-Level Factorization: First, to compute the local LMM result, in the above rule (1) [37], θ is partitioned as $\theta[1:c_{S_1}]$ and $\theta[c_{S_1}+1:c_T,]$ because the columns of T are assumed to be two *disjoint* sets from D_1 and D_2. To tackle the overlapping columns, in our modified rule (2), mapping matrix M_k brings more flexibility in choosing the columns of S_k. Second, to compute the final result, In the original rule (1), two

Algorithm 1: Linear Regression Using Gradient Descent ([37]).

```
Input: Target table $T$, label matrix $Y$, parameters $w$,
    learning rate $\gamma$
  for $i \in 1 : n$ do
    $w = w - \gamma(T^T((Tw) - Y))$
  end for
```

local LMM results (i.e., $D_1\theta[1 : c_{D_1},]$ and $D_2\theta[c_{D_1} + 1 : c_T,]$) are simply added up via indicator matrices I_1 and I_2. However, as stated, we need to handle redundancy when generalizing the LA factorization problem.

ML Model Training With Factorization: Progressing with this notion, if an ML model is built using linear operators, it can be trained directly on factorized data sources. Using the example showcased in Fig. 2, let's consider the task of training a linear regression model to predict mortality. The features for this task reside in S_1 and S_2, connected by patient names as the joinable key. To enable linear regression on factorized data sources, we must first deconstruct the training algorithm.

Algorithm 1, as an example, outlines the LA operations in training linear regression models using gradient descent. This encompasses two primary LA operations: element-wise scalar operations and matrix multiplications. With substituting T with $I_1D_1M_1^T + I_2D_2M_2^T$, the LR model can be trained over factorized data sources, where I_* and M_* are stored in the metadata catalog described in Section II-B.

Current Implementation, Extensibility and Challenges: Our running example and current implementation replace null values to zero and apply one-hot-encoding for categorical variables. Extending the implementation to replace the null values with mean/median values or user-specified values and other methods for handling categorical variables is straightforward. Based on the process we described, we showed a simple example of how Amalur uses DI metadata in factorized model training. Nonetheless, such processes might be less straightforward due to more complex schema or row mappings produced by the corresponding data integration processes. For example, consider the cases where we have $1 : n$ mappings among the schema attributes of the source tables and the one of the target table (e.g., *fullname* mapping to *first name* and *last name*), or the cases where source tables contain duplicated information (i.e., repeated entities) and require dedicated solutions. Embedding such DI metadata into factorization techniques is part of our future directions. Another interesting direction is to support more ML models, e.g., transformers [71]. In existing works, non-linearity is studied with regard to feature interactions [69] and ML models, e.g., Gaussian Mixture Models, Neural Networks [70].

B. Cost Estimation Challenge: To Factorize or to Materialize

Factorization has shown its efficacy at increasing the efficiency of model training [35], [36], [37], [38], [67], [68] However, the question of *when* to factorize is not fully answered. We illustrate the problem intuitively in Fig. 6. Let us assume that there exists a borderline (the curvy purple line), between the

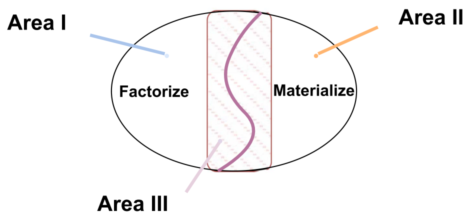


Fig. 6. Abstraction of different decision areas (factorize/materialize) and their boundaries.

cases where factorization is faster and the cases where materialization is faster. Areas I and II cover the cases when it is easy to decide on factorization or materialization, respectively, while area III covers the harder cases. The state-of-the-art solution [37] only resolves the cases in Area I, missing many potential cases in Area III where factorization is faster.

Essentially, cost estimation depends on four factors: the ML model, LA operators, hardware and underlying data. Given a model, its architecture and algebraic computations are fixed; we also know which LA operators are affected by factorization [37]. To examine the relative speedup of factorization, we mainly need to inspect data redundancy [35], [37], and the interactions between physical data transfers (e.g., network and memory bandwidth) [72].

1) Existing Solutions: We inspect data redundancy first, which is the main indicator of decision boundary in existing works. In general terms, if by joining the source tables we obtain a target table with more instance redundancy than source tables, factorization may be faster than materialization. To distinguish the Area I in Fig. 6 from others, two heuristic metrics are proposed: *tuple ratio* (TR) and *column (or feature) ratio* (FR) [37], [69], [73]. The TR, representing duplicated rows post-join, is defined as $\frac{\#tuples\, of\, S_2}{\#tuples\, of\, S_1}$. The FR quantifies the feature column proportion between two joining tables, denoted as $\frac{\#columns\, of\, S_1}{\#columns\, of\, S_2}$. Collectively, these metrics measure the data redundancy in target tables relative to source tables.

Before materialization, several parameters within the silos are crucial for understanding redundancy. These include source descriptions (e.g., the number of sources, column and row count for each source, and the null value ratio) and source correspondences (such as column and row matching between sources). Determining the boundary over separate data sources is more challenging compared to existing works over inner joins, as we need to conduct accurate cost estimation by incorporating the aforementioned DI metadata.

2) Amalur's Cost Estimator on CPUs: In the current implementation[3], we have established logic-based theoretical frameworks suitable for scenarios in Area I and II, i.e., with the existing schema mappings as early-pruning rules. We provide an example in Example 5.1. However, such schema-only rules cannot cover all the cases, as the relative comparison between target redundancy and source redundancy also depends on the row matching, i.e., Area III in Fig. 6. In Amalur, we also

[3]Amalur's CPU-based implementation is currently under submission to an anonymous conference, including methods, synthetic data generator and experiment results. Here we report the results for the completeness of our approach. Our main contribution is the GPU-based estimator in Section V-C.

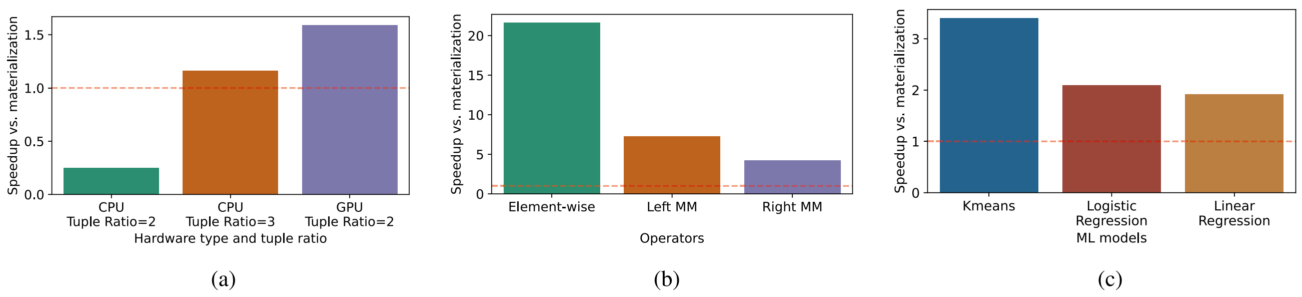


Fig. 7. (a) Speedups of factorization compared to materialization in three example configurations. Speedups are divergent with different configurations. (b) Speedups of operators and model training with factorization. Various LA operators shows significantly different speedups with factorization. (c) Speedups of operators and model training with factorization.

apply an ML-based cost estimation module that considers all three types of metadata mentioned in Section III, e.g., source and target schemata, schema mapping and row matching, ML model architecture (LA operators) and hyper-parameters, CPU memory and bandwidth.

Example 5.1: Consider Example 2 in Table I. m_1 is a full tgd, i.e., m_1 does not contain existentially quantified variables. All the attributes of target schema T come from at least one source schema. In such a case, the number of columns in T is less than or equal to the total number of columns in S_1 and S_2 participating in factorization. In the use cases of feature augmentation and VFL, the number of rows in T is usually less than or equal to the total number of rows in S_1 and S_2. That is, the materialized target table T does not contain more redundancy than the source tables. Thus, factorization will not bring performance improvement, which makes it a case in area II of Fig. 6. In similar cases, we can make a straightforward decision on choosing materialization.

Results: We evaluated our approach over approximately 1800 synthetic datasets and seven real datasets.[4] Amalur outperforms baselines [37] for estimating the faster strategy between materialization and factorization. For instance, over real datasets the state-of-the-art approach with tuple ratio and feature ratio [37] has achieved 0.464 for accuracy and 0.211 for F1 score, while Amalur achieved 0.905 and 0.821 accordingly. Factorization enhanced 31% in model training, delivering speedups of up to 4x. By identifying tasks suitable to factorization, our estimator managed to boost the system's overall performance by 20% across all ML model training.

3) Theoretical and Practical Challenges: The above example is one of the simplest applications of mapping formalism in the context of estimating performance improvement of factorization. There are more types of tgds describing more complicated dataset relationships, e.g., nested tgds [74] and plain SO tgds [75]. The discussion could also be expanded to more types of metadata, e.g., expand the existing entity resolution approaches [76] and come up with other pruning rules. In short, utilizing DI metadata in factorization needs more effort from data integration theoreticians and practitioners.

In practice, LA operators are often performed by GPUs. An estimator designed for CPUs might not be well-suited for GPU architectures due to differences in hardware configurations, including cache hierarchy, memory access speed, etc, which leads to the following challenge.

C. Cost Estimation Challenge: From CPU to GPU

The advent of LA accelerators in contemporary GPUs has substantially accelerated LA computations. This development also holds promise for enhancing factorized model training leveraging tensorized DI metadata.

We implemented three representative ML models: linear regression, logistic regression, and KMeans in both their original and factorized forms. We implemented the LA operators using CuPy[5], which is a CUDA-enabled Python numerical library. Below, we identify key insights through experiment observations in Fig. 7. Furthermore, we discuss two new challenges of cost estimation on GPUs with our initial exploration.

1) Two Insights From Experiment Observations. Hardware-related factors in cost estimation: During the development of CPU-based estimator in Section V-B, we discovered that the choice depends on three factors: data (incl. DI metadata), ML algorithms (LA operators), and hardware configuration. As discussed in Section V-B given an ML model, the data redundancy affects factorization or materialization. The heuristic metrics for measuring data redundancy are TR and FR [37]. Fig. 7(a) shows the results of training logistic regression models on CPUs using synthetic datasets. With a TR=2, factorization is slower than materialization, while with a TR=3, factorization is faster than materialization. The data-related factor, e.g., TR, is important in choosing between factorization and materialization. Now we compare the green bar (TR=2, CPU) with the purple bar (TR=2, GPU). Even with the same TR (i.e., same data redundancy), the speedup of factorization against materialization is significant. The reason is the inherent high parallelism and LA-friendly architecture of the GPU, which significantly accelerate LA operations and lead to an even more remarkable speedup even at lower TR. Thus, in complex data integration scenarios in Area III, studying hardware-related parameters, such as parallelism is imperative.

Speed-Up Over Individual LA Operators: Different combinations of LA operators present within the model training algorithm also influence the attainable speedups of factorized model training. Fig. 7(b) and (c) depict the speedups of both LA operators and ML model training. Element-wise

[4]Project Hamlet datasets: https://adalabucsd.github.io/hamlet.html
[5]https://cupy.dev/

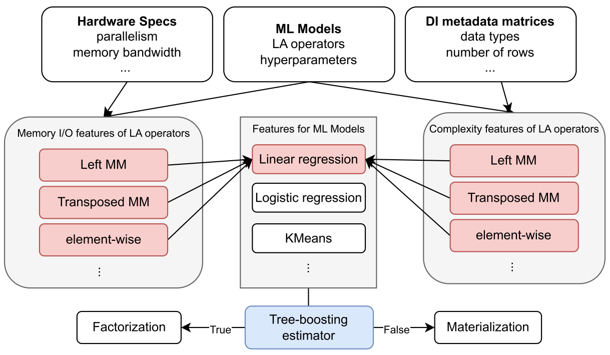


Fig. 8. Workflow of our cost estimator on GPUs.

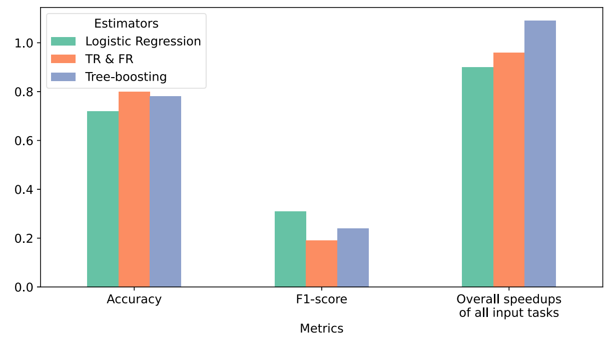


Fig. 9. Performance of Tree-boosting cost estimator compared to Logistic Regression and SoTA estimators.

scalar operators show substantially higher speedups compared to matrix multiplications, primarily because of the absence of internal data synchronization. Model training algorithms containing a larger proportion of element-wise operations can consequently achieve more significant speedups. Hence, training the KMeans algorithm using factorization demonstrates considerably higher speedups than training logistic regression.

2) Our Initial Exploration of Cost Estimation on GPUs: The cost estimation's objective is to choose between factorization and materialization. Continuing our work on cost estimation on CPUs in Section V-B, we developed an ML-based cost estimator for GPUs. Our key insight is utilizing the metadata mentioned in Section III, e.g., hardware details, model architecture, and source data characteristics, including DI metadata. To handle the non-linear relationships between these factors that influence the speedups of factorization, we adopted a *tree-boosting model*, i.e., XGBoost [77]. Tree-boosting models are particularly notable in the domain of cost estimators [78], [79] because of their explainability and fast prediction speeds.

Workflow of Amalur's estimator on GPUs: Fig. 8 shows the workflow of the tree-boosting cost estimator in Amalur. Our tree-boosting cost estimator utilizes the metadata from the catalog. Importantly, given the wide adoption of GPUs as foundational infrastructure in recent ML applications, our focus is estimating the costs of ML training on GPUs.

Furthermore, we take into account the computational complexities of LA operators, both under factorization and materialization. Considering that operators with factorization and materialization have unique complexities and memory I/O demands, we normalize these metrics based on their total amount and hardware parallelism.

Preliminary Results: Fig. 9 presents the performance metrics of our preliminary cost estimator compared to two baselines: i) an estimator applying a linear regression model in which non-linear feature interactions are neglected, and ii) the state-of-the-art empirical estimator [37], i.e., TR and FR. Our tree-boosting-based estimator demonstrates the highest overall speedups compared to the other two estimators, even though it did not achieve the top marks in accuracy and F1-score. Conversely, the empirical estimator achieves the highest accuracy but generates excessive False Positive estimations, leading to a reduced F1-score and overall speedups. While the LR estimator achieves the highest F1-score, its overall speedups are less than our estimator.

Despite the impressive overall speedups, our tree-boosting estimator correctly identified scenarios where factorization is faster in only 69% of samples. Furthermore, the average speedups for False Positive samples are even lower than those of True Negatives. This implies that certain tasks trained with factorization are significantly slower than materialization, negatively affecting the overall performance of the Amalur system. These results highlight that there remains considerable work in constructing a more accurate estimator with respect to the overall performance improvement.

3) The Road Ahead: From preliminary results, we observed that the cost estimation for factorization or materialization on GPU is a difficult problem. The remaining research gaps are more than just adding more hardware-related parameters to the previous CPU-based estimator. We are working on improving our current approach in the following two directions.

A more comprehensive study on GPU-related factors: Preliminary results indicate that GPUs can enhance the proportion of tasks where factorization proves faster. However, the real potential for speedup benefits can only be harnessed with a more accurate cost estimator. Our current estimator only uses simplistic hardware information, limiting its portability due to the absence of detailed hardware architectural characteristics. A more effective cost estimator would ideally incorporate micro-level hardware specifics.

Dynamic training of the ML-based cost estimator: Our current estimator requires pre-training before deployment. Yet, in many real-world applications [78], [79], training occurs in real-time, eliminating the need for a dedicated data preparation phase to fine-tune the estimator. Exploring this dynamic training approach is on our research horizon.

Summary & Opportunities

– DI metadata is useful for factorization over silos, and it requires much more research.

– Utilizing a GPU can notably accelerate model training with factorization, leveraging the DI metadata. This results in factorization outperforming materialization for some ML training tasks.

– Challenges remain to be explored for developing an effective and portable cost estimator on GPUs.

VI. Data Integration and Federated Learning

As indicated, various regulations govern data, e.g., GDPR and HIPAA. Consequently, it is imperative to incorporate privacy constraints into ML workflows. Privacy preservation indicates that the ML process should not reveal raw data. It necessitates an adaption of factorized learning to federated learning (FL). Current FL systems take DI as a separate preprocessing step, focusing on the learning side. In existing federated learning studies, dataset relationships are oversimplified compared to the past decades of development in the DI field. By putting DI and FL together, our vision is two-fold. First, DI metadata brings new computation and privacy-preserving opportunities in FL frameworks (Sections VI-A and VI-B). Second, new research challenges emerge when we expand the problem setting of existing FL works to more general DI scenarios (Section VI-C).

A. FL Computation in Amalur

In FL, a crucial prerequisite is establishing alignments among data silos, i.e., obtaining their column and row matching. This typically requires ML engineers to prepare a subset of local data by adding or removing feature and instance candidates from different data silos. With our proposed mapping and indicator matrices in Section IV, the subsets of local data can be represented and embedded in the federated models, which has a great potential to automate the whole process.

In what follows, we explain our intuition with the *vertical federated linear regression* (VFLR) algorithm from [34] and Example 2 in Table I. The VFLR training objective is:

$$\min_{\substack{\theta_1,\theta_2;\\ D_1,D_2}} \sum_i \left\| D_1^{(i)}\theta_1 + D_2^{(i)}\theta_2 - Y^{(i)} \right\|^2 \tag{3}$$

where D_1 and D_2 are data matrices as defined in Section IV-D, Y is the label space of D_1, θ_1, and θ_2 are the local FLR model parameters of S_1 and S_2, (i) denotes the row index of data instances in the matrix.

Factorized Learning to Federated Learning: In Amalur, we perform FL by extending our factorized learning approach, described in Section II. The performance of trained VFLR models depends on the quality of D_1 and D_2, which are prepared before and fixed during training. Refining the performance of VFLR models typically requires regenerating D_1 and D_2. With our mapping and indicator matrices, we can integrate $I_1 D_1 M_1^T$ and $I_2 D_2 M_2^T$ into the VFLR training as an end-to-end optimization procedure in a similar fashion as Section V-A.

Based on the above (3), we can extend our factorized learning approach in Section V-A to FL. We can consider the mapping matrix M_1 to match the columns of the data matrix D_1 to the respective parameters θ_1. Conversely, we use the mapping matrix M_1 to map the parameter to the data, such as if we have the VFLR model defined by parameters $\theta = (\theta_1, \theta_2)$. Thus M_1 applied to θ results in θ_1.

B. Data Privacy in Amalur With Common FL Assumptions

We first discuss common assumptions made in existing FL works for tabular data [80], [81], [82], [83]. Essentially, entity resolution (ER) and FL are often treated as two isolated tasks. ER is seen as a preprocessing step, leading to the assumptions below.

Assumption 1: The global schema is not communicated amongst the parties except the key.

Assumption 1 means that i) each party knows that they share a common key column; ii) each party does not know the feature columns of other parties. Continuing with the running example, consider the case that the table of party A has the schema $S_1(n, m, a, hr)$, while party B has the schema $S_2(n, a, o, dd)$. Through entity resolution, e.g., private set intersection (PSI), parties A and B know that they have the identifier column n (name),[6] but they do not know the rest of the columns of each other. In Amalur, we implemented the PSI step in Python using delta-psi-Py, which provides the row-matching information. With row matching, we generate indicator matrices as discussed in Section IV-B.

Assumption 2: The data is assumed to be independent in a single dimension (row or column) amongst the data sources.

As mentioned in Section II-A, existing FL implementations either handle silos with disjoint feature columns but the same set of data instances (VFL) or disjoint data instances but the same set of feature columns (HFL) [34], [80], [81], [84]. We explain Assumption 2 with the previous example of $S_1(n, m, a, hr)$ and $S_2(n, a, o, dd)$. The key column n (name) does not serve as a feature; the data sources, party A and party B, do not have overlapping feature columns in Assumption 2. This means that properties of independent computations can be used.

Data Privacy Preservation in Amalur: Continuing Section VI-A, we can ensure privacy under the above two assumptions. Because of Assumption 2, operations on the data can also be considered independent, which means the aggregation of results would heavily depend on the architecture of the chosen VFL model. This can be observed in (3) for the VFLR optimization equation for θ_1, θ_2. In (3), the results of $Y_1 = S_1\theta_1$ and $Y_2 = S_2\theta_2$ do not impact each other. Therefore, we can use latent representations L_1, L_2, which can be derived by applying specific functions, such as LA computations on S_1, S_2. For example, if we consider a neural network, the values obtained from the intermediate layers are considered latent representations. Instead of performing the model training with S_1 and S_2, the model can be trained using S_1 and latent representation L_2 [85]. That is, party B does not need to send data but latent representation to party A, which makes the training process more secure.

$$\min_{\substack{\theta_1,\theta_2;\\ I_1,M_1,L_2}} \sum_i \left\| (I_1 D_1 M_1^T)^{(i)}\theta_1 + (L_2)^{(i)}\theta_2 - Y^{(i)} \right\| \tag{4}$$

Moreover, when we consider the data integration scenarios under Assumptions 1 and 2, limited information can be learned from the training process. Thus, there will be less known about the other parties' data; even the model architecture could be unknown. For linear models, we can extend the factorization approach in Section V to FL while preserving data privacy.

[6]In this specific example, we assume that the name column is a key, i.e., no duplicated name values.

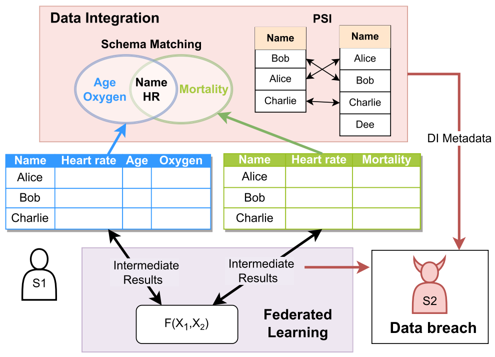


Fig. 10. VFL workflow and data breach.

C. Proposed Assumption and New Challenges

Assumptions 1 and 2 are far from real-life data integration scenarios. Data sources often have both column and row overlaps. This is why many DI pipelines have schema matching and entity resolution steps. Thus, we define and use Assumption 3 rather than the previous two assumptions.

Assumption 3: Data silos may share both overlapping feature columns and data instances.

The running example illustrates Assumption 3, where S_1 and S_2 share feature column age and the row for *Jane*. Table I shows the general data integration scenarios based on the Proposed Assumption 3. In the following, we discuss the new research opportunities based on such a more general assumption.

Improving Model Effectiveness Through Redundancy Detection: As discussed in Section IV-C, with overlapping columns and rows, two source tables may share redundant values that redundancy matrices can indicate. When such data redundancy is non-negligible, it may deteriorate the model's accuracy. With the redundancy matrices, it is possible to improve the data quality by detecting and eliminating such redundancy, improving the model's effectiveness. The redundancy matrices only point out the locations of the repeated values without leaking the raw values of source datasets in silos.

Privacy Concern Due to Overlapped Data: Under the Proposed Assumption 3, we revisit (3). Since parties A and B share common features, their parameters can now be partitioned into the following set: $(\theta_1, \theta_2, \theta_{1\cap 2})$. The shared parameters, i.e., $\theta_{1\cap 2}$, lead to privacy concerns, such as learning the feature values through the gradient of the shared parameters $\theta_{1\cap 2}$ [81], [86]. An example of the overlap can be seen in Fig. 10. Thus, an interesting question is: *how to utilize DI metadata for FL to avoid such privacy threats?*

Choosing Privacy-Preserving Technique: There are a number of widely used privacy-preserving techniques that are suitable for different ML algorithms and scenarios, e.g., homomorphic encryption (HE) [87], [88], secret sharing (SS) [89], [90] and differential privacy [91]. HE is typically used to encrypt sensitive information exchanged between parties. It guarantees privacy but incurs high computational cost. SS enables participating parties of FL to collaboratively compute a function while preventing all parties from accessing the input data of that function and hence preserves the privacy of input data. However, secret sharing brings about high communication cost. Differential privacy protects data privacy by adding noise to the original data or information exchanged between parties. It is more efficient than HE and SS, but may jeopardize the performance of ML models. It is an open question: How do we choose the appropriate technique when considering source dataset characteristics and their relationships?

Summary & Opportunities

– DI metadata is promising for improving ML model accuracy and automating data transformation pipelines and FL.
– More complicated dataset relationships bring more practical FL use cases but also more challenges for preserving privacy.

VII. Conclusion and Future Direction

In this work, we have explored the possibilities of bringing data integration and machine learning together. Toward this direction, we have proposed a data integration-aware ML system Amalur, which supports machine learning training and inference over silos. We have inspected the promising challenges of representing DI metadata and utilizing it for factorized and federated learning. We envision this work as one of the first steps towards bridging the recent advances in machine learning with the well-studied traditional data integration field.

References

[1] J. M. Bos et al., "Prediction of clinically relevant adverse drug events in surgical patients," *PLoS One*, vol. 13, no. 8, 2018, Art. no. e0201645.
[2] A. Hard et al., "Federated learning for mobile keyboard prediction," 2018, *arXiv: 1811.03604*.
[3] A. Doan, A. Halevy, and Z. Ives, *Principles of Data Integration*, Amsterdam, The Netherlands: Elsevier, 2012.
[4] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.
[5] R. Fagin et al., "Clio: Schema mapping creation and data exchange," in *Conceptual Modeling: Foundations and Applications*, Berlin, Germany: Springer, 2009, pp. 198–236.
[6] D. G. Brizan and A. U. Tansel, "A survey of entity resolution and record linkage methodologies," *Commun. IIMA*, vol. 6, no. 3, 2006, Art. no. 5.
[7] N. Makrynioti and V. Vassalos, "Declarative data analytics: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 6, pp. 2392–2411, Jun. 2021.
[8] X. Zhou, C. Chai, G. Li, and J. Sun, "Database meets artificial intelligence: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 3, pp. 1096–1116, Mar. 2022.
[9] S. Maximilian, O. Dan, Abo-Khamis, and N. XuanLong, "Learning models over relational data: A brief tutorial," in *Proc. Int. Conf. Scalable Uncertainty Manage.*, Cham: Springer, 2019, pp. 423–432.
[10] A. Alserafi, A. Abelló, O. Romero, and T. Calders, "Keeping the data lake in form: Proximity mining for pre-filtering schema matching," *ACM Trans. Inf. Syst.*, vol. 38, no. 3, pp. 26:1–26:30, 2020.
[11] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1335–1349.
[12] H. Köpcke, A. Thor, and E. Rahm, "Evaluation of entity resolution approaches on real-world match problems," in *Proc. VLDB Endowment*, vol. 3, pp. 484–493, 2010.
[13] S. Das et al., "Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2017, pp. 1431–1446.
[14] Z. Wang, B. Sisman, H. Wei, X. L. Dong, and S. Ji, "CorDEL: A contrastive deep learning approach for entity linkage," in *Proc. IEEE Int. Conf. Data Mining*, 2020, pp. 1322–1327.
[15] S. Krishnan et al., "ActiveClean: Interactive data cleaning for statistical modeling," in *Proc. VLDB Endowment*, vol. 9, pp. 948–959, 2016.

[16] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu, "BoostClean: Automated error detection and repair for machine learning," 2017, *arXiv: 1711.01299*.

[17] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "CleanML: A study for evaluating the impact of data cleaning on ML classification tasks," in *Proc. IEEE 37th Int. Conf. Data Eng.*, 2021, pp. 13–24.

[18] X. L. Dong and T. Rekatsinas, "Data integration and machine learning: A natural synergy," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2018, pp. 1645–1650.

[19] A. Y. Halevy, A. Rajaraman, and J. J. Ordille, "Data integration: The teenage years," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, pp. 9–16.

[20] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*, San Rafael, CA, USA: Morgan & Claypool, 2019.

[21] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, vol. 10, 1st ed. Cham, Switzerland: Springer, 2017, pp. 10–5555.

[22] D. of health and human services, "Summary of the HIPAA privacy rule - hhs.gov," [Online]. Available: https://www.hhs.gov/sites/default/files/ocr/privacy/hipaa/understanding/summary/privacysummary.pdf

[23] Information privacy act 2009. Accessed: Aug. 31, 2023. [Online]. Available: https://www.legislation.qld.gov.au/view/pdf/inforce/current/act-2009--014

[24] Personal information protection law of the People's Republic of China. Accessed: Aug. 31, 2023. [Online]. Available: http://en.npc.gov.cn.cdurl.cn/2021--12/29/c_694559.htm

[25] M. Lenzerini, "Data integration: A theoretical perspective," in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, 2002, pp. 233–246.

[26] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W.-C. Tan, "Data integration: After the teenage years," in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, 2017, pp. 101–106.

[27] W. Wang et al., "Rafiki: Machine learning as an analytics service system," 2018, *arXiv: 1804.06087*.

[28] P. Kraft, D. Kang, D. Narayanan, S. Palkar, P. Bailis, and M. Zaharia, "Willump: A statistically-aware end-to-end optimizer for machine learning inference," in *Proc. Int. Conf. Mach. Learn. Syst.*, 2020, pp. 147–159.

[29] C. Beeri and M. Y. Vardi, "A proof procedure for data dependencies," *J. ACM*, vol. 31, no. 4, pp. 718–741, 1984.

[30] R. Fagin, *Tuple-Generating Dependencies*, Boston, MA, USA: Springer, 2009, pp. 3201–3202.

[31] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger, "ARDA: Automatic relational data augmentation for machine learning," in *Proc. VLDB Endowment*, vol. 13, no. 9, pp. 1373–1387, 2020.

[32] M. Esmailoghli, J.-A. Quiané-Ruiz, and Z. Abedjan, "COCOA: Correlation coefficient-aware data augmentation," in *Proc. 24th Int. Conf. Extending Database Technol.*, 2021, pp. 331–336.

[33] A. Kumar et al., "To join or not to join? Thinking twice about joins before feature selection," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 19–34.

[34] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, Jan. 2019, Art. no. 12.

[35] M. Schleich, D. Olteanu, and R. Ciucanu, "Learning linear regression models over factorized joins," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 3–18.

[36] A. Kumar, J. Naughton, and J. M. Patel, "Learning generalized linear models over normalized data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1969–1984.

[37] L. Chen, A. Kumar, J. Naughton, and J. M. Patel, "Towards linear algebra over normalized data," in *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1214–1225, 2017.

[38] R. Alotaibi, B. Cautis, A. Deutsch, and I. Manolescu, "HADAD: A lightweight approach for optimizing hybrid complex analytics queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2021, pp. 23–35.

[39] R. Hai, S. Geisler, and C. Quix, "Constance: An intelligent data lake system," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 2097–2100.

[40] Z. Li et al., "Macaroni: Crawling and enriching metadata from public model zoos," in *Proc. Int. Conf. Web Eng.*, Springer, 2023, pp. 376–380.

[41] C. Quix, R. Hai, and I. Vatov, "Metadata extraction and management in data lakes with GEMMS," *Complex Syst. Inform. Model. Quart.*, no. 9, pp. 67–83, 2016.

[42] M. Scannapieco et al., "Privacy preserving schema and data matching," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 653–664.

[43] P. A. Bernstein, "Applying model management to classical meta data problems," in *Proc. Int. Conf. Innov. Data Syst. Res.*, 2003, pp. 209–220.

[44] P. G. Kolaitis, "Schema mappings, data exchange, and metadata management," in *Proc. 24th ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, Baltimore, MD,USA, 2005, pp. 61–75.

[45] A. Silberschatz, H. F. Korth, and S. Sudarshan, "Database system concepts," 2011.

[46] G. Singh et al., "A metadata catalog service for data intensive applications," in *Proc. ACM/IEEE Conf. Supercomputing*, 2003, pp. 33–33.

[47] A. Y. Halevy et al., "Managing Google's data lake: An overview of the Goods system," *IEEE Data Eng. Bull.*, vol. 39, no. 3, pp. 5–14, Mar. 2016.

[48] C. Koutras et al., "Valentine: Evaluating matching techniques for dataset discovery," in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 468–479.

[49] R. Hai, C. Koutras, C. Quix, and M. Jarke, "Data lakes: A survey of functions and systems," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12571–12590, Dec. 2023.

[50] S. Bharadwaj, P. Gupta, R. Bhagwan, and S. Guha, "Discovering related data at scale," in *Proc. VLDB Endowment*, vol. 14, no. 8, pp. 1392–1400, 2021.

[51] A. Khatiwada, R. Shraga, and R. J. Miller, "DIALITE: Discover, align and integrate open data tables," in *Proc. Int. Conf. Manage. Data*, 2023, pp. 187–190.

[52] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3320–3328.

[53] H. Iida, D. Thai, V. Manjunatha, and M. Iyyer, "TABBIE: Pretrained representations of tabular data," 2021, *arXiv:2105.02584*.

[54] R. Levin et al., "Transfer learning with deep tabular models," in *Proc. Int. Conf. Learn. Representations*, 2022.

[55] P. Michał et al., "STable: Table generation framework for encoder-decoder models," in *Proc. NeurIPS 1st Table Representation Workshop*, 2022.

[56] C. Renggli, X. Yao, L. Kolar, L. Rimanic, A. Klimovic, and C. Zhang, "SHiFT: An efficient, flexible search engine for transfer learning," in *Proc. VLDB Endowment*, vol. 16, no. 2, pp. 304–316, 2022.

[57] I. Trummer and C. Koch, "Approximation schemes for many-objective query optimization," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1299–1310.

[58] C. Coleman et al., "DAWNBench: An end-to-end deep learning benchmark and competition," *Training*, vol. 100, no. 101, 2017, Art. no. 102.

[59] A. Sabne, "XLA: Compiling machine learning for peak performance," 2020.

[60] H. Vanholder, "Efficient inference with tensorrt," in *Proc. GPU Technol. Conf.*, 2016, Art. no. 2.

[61] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[62] D. He et al., "Query processing on tensor computation runtimes," in *Proc. VLDB Endowment*, vol. 15, no. 11, pp. 2811–2825, 2022.

[63] Y.-C. Hu, Y. L. Li, and H.-W. Tseng, "TCUDB: Accelerating database with tensor processors," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2022, pp. 1360–1374.

[64] M. Kim and K. S. Candan, "TensorDB: In-database tensor manipulation with tensor-relational query plans," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2014, pp. 2039–2041.

[65] D. Koutsoukos et al., "Tensors: An abstraction for general data processing," in *Proc. VLDB Endowment*, vol. 14, no. 10, pp. 1797–1804, 2021.

[66] W. Sun, A. Katsifodimos, and R. Hai, "An empirical performance comparison between matrix multiplication join and hash join on GPUs," in *Proc. IEEE Int. Conf. Data Eng. Workshop*, 2023, pp. 184–190.

[67] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich, "AC/DC: In-Database learning thunderstruck," in *Proc. 2nd Workshop Data Manag. End-To-End Mach. Learn.*, Houston TX USA, 2018, pp. 1–10.

[68] D. Justo, S. Yi, L. Stadler, N. Polikarpova, and A. Kumar, "Towards a polyglot framework for factorized ML," in *Proc. VLDB Endowment*, vol. 14, no. 12, pp. 2918–2931, 2021. [Online]. Available: https://doi.org/10.14778/3476311.3476372

[69] S. Li, L. Chen, and A. Kumar, "Enabling and optimizing non-linear feature interactions in factorized linear Algebra," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2019, pp. 1571–1588.

[70] Z. Cheng, N. Koudas, Z. Zhang, and X. Yu, "Efficient construction of nonlinear models over normalized data," in *Proc. IEEE 37th Int. Conf. Data Eng.*, 2021, pp. 1140–1151.

[71] V. Ashish et al., "Attention is all you need," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[72] M. Zhao et al., "Understanding data storage and ingestion for large-scale deep recommendation model training," in *Proc. 49th Annu. Int. Symp. Comput. Architecture*, 2022, pp. 1042–1057.

[73] Z. Cheng, N. Koudas, Z. Zhang, and X. Yu, "Efficient construction of nonlinear models over normalized data," in *Proc. IEEE 37th Int. Conf. Data Eng.*, 2021, pp. 1140–1151.
[74] P. G. Kolaitis, E. Sallinger, and V. Savenkov, "On the language of nested tuple generating dependencies," *ACM Trans. Database Syst.*, vol. 45, no. 2, pp. 1–59, 2020.
[75] M. Arenas et al., "The language of plain SO-tgds: Composition, inversion and structural properties," *J. Comput. Syst. Sci.*, vol. 79, no. 6, pp. 763–784, 2013.
[76] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis, "An overview of end-to-end entity resolution for big data," *ACM Comput. Surv.*, vol. 53, no. 6, pp. 1–42, 2020.
[77] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785
[78] T. Chen et al., "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Conf. Operating Syst. Des. Implementation*, 2018, pp. 578–594.
[79] A. Adams et al., "Learning to optimize halide with tree search and random programs," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 121:1–121:12, 2019.
[80] K. Cheng et al., "SecureBoost: A lossless federated learning framework," *IEEE Intell. Syst.*, vol. 36, no. 6, pp. 87–98, Nov./Dec. 2021.
[81] F. Fu, H. Xue, Y. Cheng, Y. Tao, and B. Cui, "BlindFL: Vertical federated machine learning without peeking into your data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2022, pp. 1316–1330.
[82] F. Fu et al., "VF ^{2} Boost: Very fast vertical federated gradient boosting for cross-enterprise learning," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 563–576.
[83] W. Fang et al., "Large-scale secure XGB for vertical federated learning," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2021, pp. 443–452.
[84] D. Liu, L. Bai, T. Yu, and A. Zhang, "Towards method of horizontal federated learning: A survey," in *Proc. 8th Int. Conf. Big Data Inf. Analytics*, 2022, pp. 259–266.
[85] C. Thapa, M. A. P. Chamikara, and S. Camtepe, "SplitFed: When federated learning meets split learning," 2020, *arXiv: 2004.12088*, [Online]. Available: https://arxiv.org/abs/2004.12088
[86] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 3–18.
[87] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *EURASIP J. Inf. Secur.*, vol. 2007, 2007, Art. no. 013801.
[88] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, Springer, 1999, pp. 223–238.
[89] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[90] A. Beimel, "Secret-sharing schemes: A survey," in *Proc. Int. Conf. Coding Cryptol.*, Springer, 2011, pp. 11–46.
[91] C. Dwork, "Differential privacy: A survey of results," in *Proc. Int. Conf. Theory Appl. Models Computation*, Springer, 2008, pp. 1–19.

Ziyu Li received the BS degree from the South China University of Technology, China, in 2017, and the MS degree in computer science from the Delft University of Technology, The Netherlands, in 2019. She is currently working toward the PhD degree with the Department of Software Technology, Delft University of Technology, The Netherlands. Her research interests include data management, machine learning, and metadata management.

Wenbo Sun received the master's degree in computational science from the University of Amsterdam. He is currently working toward the PhD degree with TU Delft. Prior to joining TU Delft, he worked as a software engineer. His research focuses on database system over modern hardware and performance engineering.

Danning Zhan received the bachelor's degree in mathematics, in 2020, and the master's degree in computer science, in 2022. He is currently working toward the PhD degree with the Department of Software Technology, TU Delft. His research explores data privacy in machine learning on distributed data.

Yan Kang is currently a researcher with the AI Department of WeBank, Shenzhen, China. His works focus on the research and implementation of privacy-preserving machine learning and federated learning. His research was authored or coauthored in well-known conferences and journals, including IEEE ISP, IEEE TBD, IJCAI, ICDE, and IEEE TKDE.

Lydia Chen is a professor with the University of Neuchatel and TU Delft. She is the director of Distributed Learning Systems Lab. She returned to academia, after a decade of industry experience with the IBM Research Zurich Lab. Her research interests lie in the distinct areas of deep machine learning, big data systems, and privacy enhancing technology. She is also the co-founder of BlueGen.ai. Her research is supported by the Swiss National Science Foundation, Dutch National Science Foundation the European Union, IBM Research, ABB, TU Delft, Aegon, Tata Steel, and ASML.

Alessandro Bozzon received the PhD degree from Politecnico di Milano, in 2009, with a thesis focused on model driven approaches for the design, development and automatic code generation of Search Based Applications. He is professor of Human-Centered Artificial Intelligence and head of the Department of Sustainable Design Engineering, Delft University of Technology. He is principal investigator of Urban Data and Intelligence at the Amsterdam Institute for Advanced Metropolitan Solutions (AMS), member of the Steering Committee of the International Conference of Web Engineering (ICWE), and member of the Steering Committee of the Human Computation and Crowdsourcing (HCOMP) conference. His research lies at the intersection of human-computer interaction and machine learning. He co-authored more than 200 papers in leading peer-reviewed international journals and conferences, where he also regularly serves as senior program committee member.

Rihan Hai received the PhD degree from RWTH Aachen University, Germany. She is an assistant professor with the Web Information Systems Group, Delft University of Technology, The Netherlands. Her research focuses on data lakes, data integration, and data management for machine learning. She has served as a program committee member of database conferences, such as VLDB, ICDE, and EDBT, and a journal reviewer of the *IEEE Transactions on Knowledge and Data Engineering*, *VLDB Journal*, *Journal of Machine Learning Research*, and *IEEE Transactions on Parallel and Distributed Systems*.