Amalur

Data Integration Meets Machine Learning

Hai, Rihan; Koutras, Christos; Ionescu, Andra; Li, Ziyu; Sun, Wenbo; van Schijndel, Jessie ; Kang, Yan; Katsifodimos, Asterios

DOI
10.1109/ICDE55515.2023.00301

Publication date
2023

Document Version
Final published version

Published in
Proceedings of the 2023 IEEE 39th International Conference on Data Engineering, ICDE 2023

Citation (APA)
Hai, R., Koutras, C., Ionescu, A., Li, Z., Sun, W., van Schijndel, J., Kang, Y., & Katsifodimos, A. (2023). Amalur: Data Integration Meets Machine Learning. In *Proceedings of the 2023 IEEE 39th International Conference on Data Engineering, ICDE 2023* (pp. 3729-3739). (Proceedings - International Conference on Data Engineering; Vol. 2023-April). IEEE. https://doi.org/10.1109/ICDE55515.2023.00301

2023 IEEE 39th International Conference on Data Engineering (ICDE) | 979-8-3503-2227-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDE55515.2023.00301

Amalur: Data Integration Meets Machine Learning

Rihan Hai Christos Koutras Andra Ionescu Ziyu Li Wenbo Sun
Jessie van Schijndel Yan Kang* Asterios Katsifodimos
Delft University of Technology, *WeBank
{initial.lastname}@tudelft.nl, yangkang@webank.com

***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 and security 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 how to bridge the traditional data integration (DI) techniques with the requirements of modern machine learning. We explore the possibilities of utilizing metadata obtained from data integration processes for improving the effectiveness and efficiency of ML models. Towards this direction, we analyze two common use cases over data silos, feature augmentation and federated learning. Bringing data integration and machine learning together, we highlight new research opportunities from the aspects of systems, representations, factorized learning and 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. Take, for instance, drug risk prediction: the features can reside in datasets collected from clinics, hospitals, pharmacies, and laboratories distributed geographically [1]. Another example is training models for keyboard stroke prediction: training requires data from millions of phones [2].

Data integration 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) [6]. Yet, a data integration system's goal is to facilitate query answering or data transformation over silos, and not to directly support machine learning applications. As a result, practitioners nowadays tackle silos with DI systems and ML tooling separately, as shown in the following.

Running example. Consider the feature augmentation example in Figure 2, where the downstream ML task is to predict the mortality (binary classification) of patients 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 feature columns a (age) and hr $(resting\ heart\ rate)$. To improve the model's accuracy,

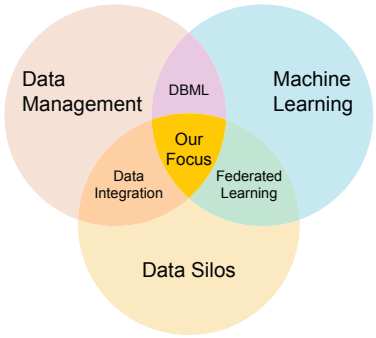

Figure 1: Scope of this line of work

a data discovery system is employed to discover a related table S_2*(m,n,a,o,dd)* (Figure 2b), 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. Figure 1 illustrates our problem scope. Recent advances of *in-database machine learning* [7]–[9], mainly consider a single database instead of data silos[1]. Traditional data integration solves the data management issues of data silos [19]. In a similar way, federated learning (FL) studies machine learning with training data residing in data silos [20]. In this paper, we argue that when data management, data silos, and machine learning meet, there is a new set of challenges and opportunities for research and optimization.

Issues with the separation of DI and ML. As shown in Figure 2c-d, to use the data from the two tables S_1 and S_2, a data scientist would need to rely on a data integration system, or else manually find the schema mapping and entity resolution between the two given tables. We elaborate on the explanation of schema mappings in Section III-A. Then, a data integration system can integrate these source tables by merging the mapped columns and linked entities (i.e., matched rows).

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

2375-026X/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDE55515.2023.00301

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. Meanwhile, it assumes that users are aware of the data sources, know the principles of data integration, and/or are familiar with DI tools. In other words, there is great potential to utilize DI techniques to reduce the human burden and automate ML pipelines.

Research vision & question. Data integration is a well-studied research area with mature logic-based theoretical frameworks, techniques, and systems [19], [21], [22]. We envision novel systems that combine DI techniques (schema matching, schema mapping, entity resolution, query reformulation, etc.) and ML pipelines (e.g., feature selection and augmentation, model training), while also expanding to more ML philosophies (e.g., federated learning). As the starting point, we ask a fundamental question:

Q: Can we use data integration metadata to improve the effectiveness and efficiency of ML model training?

With this line of work, we aim to investigate whether the metadata obtained from data integration, specifically the output of *schema matching* and *entity resolution*, can benefit downstream ML tasks. To this end, we present new research challenges that arise when we design a novel data integration system, which extends the concepts of query rewriting and data transformations for the needs of ML model training, and saves costs associated to intermediate result materialization and the exporting of target tables.

In this paper we focus on these challenges in four aspects: *system-design*, *metadata representation*, *performance*, and *privacy*. The contributions of this paper go as follows:

- *System-design (Section II).* We demonstrate the design of *Amalur*, a novel data integration system, which supports end-to-end, scalable machine learning pipelines over data silos. We elaborate on the research challenges of building such a system.
- *Metadata representations (Section III).* We propose matrix-based representations for data integration 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.
- *Performance. (Section IV)* We highlight the new opportunities for utilizing DI metadata to improve the time-wise efficiency of ML model training over data silos.
- *Privacy (Section V).* We discuss the research challenges of improving vertical federated learning with data integration metadata.

II. Amalur: An ML-oriented data integration system

In this section, we introduce our proposed system Amalur. We explain the challenges of two common ML use cases, namely feature augmentation and federated learning, and discuss how Amalur can tackle these challenges.

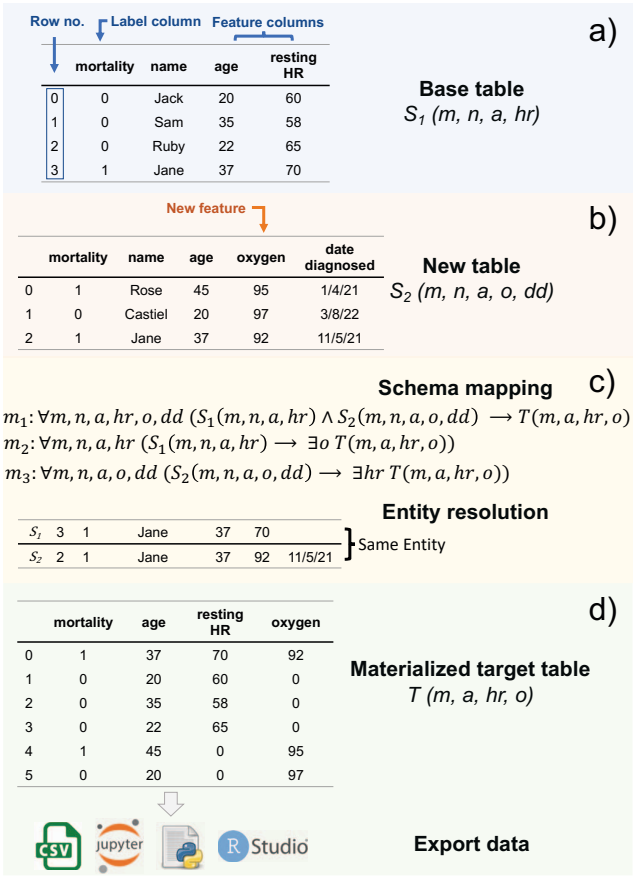

Figure 2: Traditional integration of data silos for ML

A. Amalur overview

We are currently developing *Amalur*, a *model lake* system that is based on our work on data lakes [23] and model zoos [24]. With DI metadata, Amalur features solving the challenges of scalable training of ML models over data silos, reducing the manual work of integrating the data and speeding up model training. Figure 3 provides a high-level overview of Amalur with key components relevant to this paper.

User inputs. Amalur allows users (e.g., physicians, data scientists) to train models on data silos. The user may already have an ML model, e.g., defined in Python scripts. There might also be constraints specific to a user and silos, e.g., data privacy regulations such as GDPR [25].

Hybrid metadata catalog. One of the fundamental components of Amalur is the metadata catalog. It stores the metadata of data and ML models. Data-related metadata includes the basic metadata and data integration metadata. Collected from the silos, the basic metadata describes each data source, e.g., source table schema, data types, integrity constraints, data provenance information such as silo location. In this work, by *data integration metadata* we refer to the metadata generated during the data integration process, e.g., column relationships from schema matching and row match-

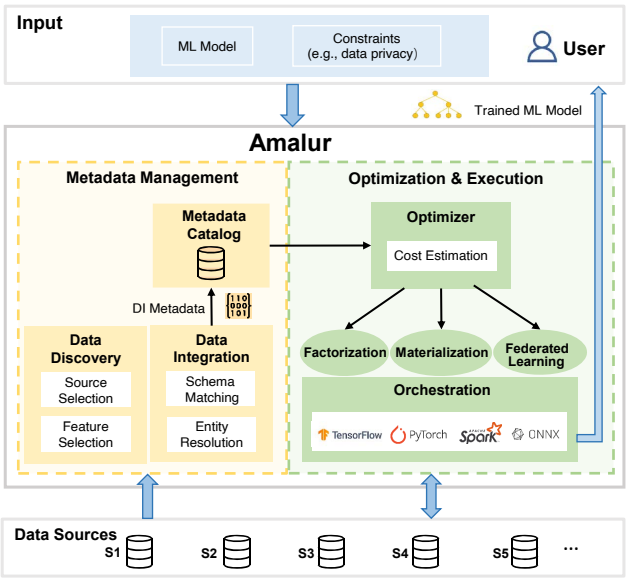

Figure 3: An overview of Amalur

ing from entity resolution. Model-related metadata include the model execution environment, configurations (e.g., hyper-parameters), input/output (e.g., prediction class), evaluation performance (e.g., model accuracy), etc. The metadata catalog also keeps track of the connections between the model and its training datasets. We have addressed the representations of basic metadata of source tables [26] and ML models [24]. In this work, we focus on data integration metadata and explain their matrix-based representation in Section III.

Optimization and coordination. Another essential component is the optimizer. Given the input ML model, constraints and metadata from the metadata catalog, the optimizer estimates the cost and decides how the ML model training will be performed over silos: 1) the ML model is decomposed and the computation is pushed down to the source tables stored in the silos, i.e., factorization; 2) the source tables are joined and the generated target table is exported for model training, i.e., materialization; 3) the learning process is split given privacy-preservation constraints, i.e., federated learning. We elaborate on the details in Section IV and Section V.

Compilation, orchestration and distribution. Finally, the execution plan from the optimizer is compiled into concrete programs according to the execution environment (e.g., TensorFlow, PyTorch, Spark, ONNX). The compiled binaries are executed either in a central orchestrator or multiple remote orchestrators. Specifically, for factorized executions, the executables are shipped to the different data silos and return results needed in a centralized computation.

B. Silos problem: ML use cases and DI formulation

In Table I, we provide two representative ML use cases where training data could come from silos, i.e., feature augmentation and federated learning. Existing solutions for factorization over joins [27]–[29] mostly tackle inner joins. In this work, we also deal with left joins, full outer joins, and unions. As shown in Table I, for the use cases of feature augmentation and federated learning (and possibly many more), the dataset relationships between source tables and desired target table, can be captured by a class of well-studied data dependencies, i.e., tuple-generating dependencies (tgds) [30], [31], which are the commonly used formalism in data integration studies.

Use case 1: Feature augmentation is the exploratory process of finding new datasets and selecting features that help improve the ML model performance [32]–[34]. Figure 2b 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 $(oxygen)$.

Use case 2: Federated learning [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 problem setting of virtual data integration [3], FL assumes that the source data is not collected and stored at a central data store but stays at the local data stores. 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) [35]. For VFL, data sources share the overlapping data instances, but the feature columns partially overlap or not. 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 Figure 2 and Example III.1. It can also be seen as a general case of federated learning, where sources have similar schemas, and data instances (entities) that may, or may not overlap with each other.

Example 2 (inner join) represents the DI 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 use case, 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. Thus, when adding features from the new table S_2, only rows overlapped with S_1 will be selected. Example 3 can be used to describe the VFL cases where 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 schemas of S_1 and S_2 such that they share the same set of feature columns which are mapped to the target schema T. Example

Table I: Four example data integration scenarios for feature augmentation and federated learning

No.	Dataset Relationship	Schema mappings	Example use cases
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, ...

4 can represent the scenario when a new table is selected to bring more data samples. Alternatively, it can describe the HFL scenario where data sources share feature columns but not data samples.

C. Amalur workflows for the two ML use cases

Amalur workflow for feature augmentation. When there is no privacy constraint, Amalur determines the computation mechanism based on cost estimation. If the computation is performed in a factorized manner, the model is decomposed and pushed down to silos. If the computation is performed in a materialized manner, Amalur will integrate the source datasets and generate the target table.

Amalur workflow for federated learning. In the existence of privacy constraints, Amalur will conduct privacy-preserving data integration operations over the silos [36], and split the learning process over the silos. The central orchestrator will coordinate communication between silos, and the encryption/decryption during aggregating the results and updating the weights.

III. Representation: A Tale of Three Matrices

In this section, we discuss the representations that capture the metadata of data integration, which enable optimizing ML tasks. When combining the functions of data integration and machine learning in one system such as Amalur, one core task is how to represent the DI metadata. By DI metadata, we refer to the information that describes *i*) the selected data sources, e.g., table schemas and the number of rows, and the number of selected sources; *ii*) the relevance and overlap between data sources, e.g., column matching between source tables (schema matching), schema-level correspondences between source tables and the target table (schema mapping), and row matching between source tables (entity resolution). 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 III-A), *indicator matrix* for row matching (Section III-B), and *redundancy matrix* for data redundancy (Section III-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 IV. Finally, in Section III-D, we inspect the implementation of such matrix-based representations at the physical level.

Table II: 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

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)* [30], [31], which are also known as Global-Local-as-View (GLAV) assertions [21]. 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 $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over the target schema $\mathbf{T}$.

Example III.1. In Figure 2c, 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 linear algebra. To embed schema mappings in an end-to-end ML pipeline, we need a novel representation for schema mappings,

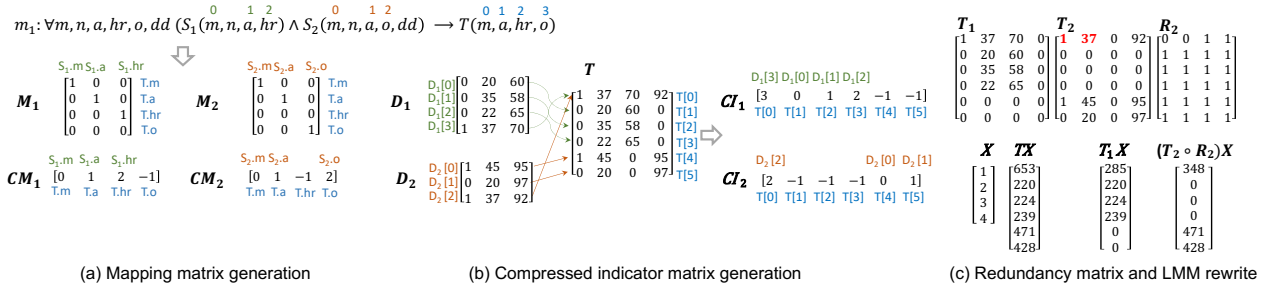

Figure 4: Mapping, indicator, and redundancy matrices of the running example

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 Figure 4a. In Table II we summarize the notations used in this paper. We abuse the notation a little and refer to both a target/source table name and its schema with T/S_k.

Definition III.1 (*Mapping matrix*)**.** Mapping matrices between source tables $S_1, S_2, ..., S_n$ and target table T are a set of binary matrices $\mathbf{M} = \{M_1, ..., M_n\}$. M_k $(k \in [1, n])$ is a matrix with the shape $c_T \times c_{S_k}$, where

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

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. Figure 4a 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. Because each attribute in the source table S_k is mapped to only one attribute in T. Thus, in each row of M_k at most one element is 1, while the rest are 0. Moreover, if an attribute of T does not have a mapped attribute in S_k, the corresponding row of the mapping matrix has only values of 0. For example, $T.o$ (column ID: 3) does not have a mapped column in S_1, thus, the last row of M_1 has only zeros, i.e., $M_1[3] = [0, 0, 0]$. To solve the sparsity problem we apply a more compressed form of mapping matrices as follows.

Definition III.2 (*Compressed mapping matrix*)**.** Compressed mapping matrices between source tables $S_1, S_2, ..., S_n$ and target table T are a set of row vectors $\mathbf{CM} = \{CM_1, ..., CM_n\}$. CM_k $(k \in [1, n])$ is a row vector of size c_T, where

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

We continue with the running example. Figure 4a 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* [27] (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 III.3 (*Compressed indicator matrix*)**.** Compressed indicator matrices between source tables $S_1, S_2, ..., S_n$ and target table T are a set of row vectors $\mathbf{CI} = \{CI_1, ..., CI_n\}$. CI_k $(k \in [1, n])$ is a row vector of size r_T, where

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

Notably, for the downstream ML algorithms, not all but only partial data of an original source table will participate in the computation as features or labels. Thus, we transform the original tables S_1 and S_2 in Figure 2a-b to their matrix forms D_1 and D_2 in Figure 4b, which only include the mapped columns. Figure 4b shows the row matching of the running example and the compressed indicator matrices, CI_1 for S_1 and CI_2 for S_2.

C. Redundancy matrix

Data integration systems often need to handle data redundancy when multiple sources have overlapping values. Consider the example in Figure 2, when a user query asks 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.

$$T_k = I_k D_k M_k^T$$

Figure 4c 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 Figure 4b-c. This is why we need the redundancy matrix, which is defined below.

Definition III.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

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

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*, which has all the element values equal to one. Figure 4c shows R_2 for S_2 given the running example, 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 data integration processes with linear algebra. Aside from the intuitive 2-dimensional 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 data integration metadata with a more expressive data structure compatible with tensor algebra and recent advances in data processing [37]–[43].

As the fundamental language of machine learning, tensor algebra and the benefits it brings in efficiency [37]–[39] have attracted the considerable attention of the ML and DB research communities. Many recent works [40]–[43] 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.

Furthermore, as dedicated tensor processing modules keep emerging, the tensor representation exhibits compatibility with new hardware. Google TPUs [44] and recent Nvidia GPUs [45] have built-in tensor cores where matrix multiplication can be computed in one single clock cycle, improving parallelism from the array-level of SIMD instructions to the matrix-level. The High Performance Computing community [46], [47] has started optimizing numeric algorithms for tensor cores. It is foreseen that the tensor-represented data processing can be significantly accelerated with co-evolving tensor algorithms, runtime, and dedicated tensor computing hardware.

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.

IV. Algebraic computation over silos

In this section, we dive into the research opportunities of conducting arithmetic computations over silos with data integration metadata. In the following, we first discuss the new challenges of generalizing the existing factorization techniques from a single database to data silos (Section IV-A), and cost estimation for choosing factorization or materialization (Section IV-B).

Factorization vs. materialization. The training process of an ML model requires complex arithmetic computations. Similar to *data warehousing* and *virtual data integration*[2], these computations during model training can be conducted in a materialized or factorized manner. Materialization requires joining the source tables and obtaining the instances of the target table before exporting it for model training, as depicted in Figure 2. Another option is *learning over factorized joins* [50], also known as *factorized learning* [51]. 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 [27]–[29], [50]–[57]. Notably, similar to traditional DI systems, materialization is not possible in some cases due to privacy constraints and other reasons, which we address in Section V. In this section, we focus on the performance implications of these two strategies.

[2]We can classify the architectures of most traditional data integration systems as *data warehousing* or *virtual data integration* [3]. In data warehousing, the target table is materialized and the materialization often requires an extract, transform, load (ETL) process. In a virtual data integration system, the target table is not materialized, and the user can pose queries against the target schema, also known as *global schema* or *mediated schema*. The user query needs to be rewritten according to the underlying source schemas, e.g., view-based query rewriting [48], [49].

A. Computation challenge: DI metadata for factorization

In the following, we explain how the data integration metadata can be used to generalize existing factorization techniques over silos (cf. Table I), and the new challenges. The existing factorization either tackles the model as a whole [50], [51] or at the linear algebra level [27], for linear or non-linear models [27]–[29], [51]–[56]. To simplify the discussion, here we use the example of LA operator *left matrix multiplication (LMM)* and its rewrite rule from [27].

New algebraic rewriting rules. Given a matrix X with the size $c_T \times c_X$, the LMM of T and X is denoted as TX. 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 [27] with our proposed rule.

$$TX \rightarrow I_1(D_1X[1:c_{S_1},]) + I_2(D_2X[c_{S_1}+1:c_T,]) \text{ [27]} \quad (1)$$

$$\Downarrow$$

$$TX \rightarrow \ I_1D_1M_1^TX \ + \ ((I_2D_2M_2^T) \circ R_2)X \text{ [Amalur]} \quad (2)$$

① *Local result generation.* We first compute $I_kD_kM_k^T$ for each source table. In this step, to reduce computation overhead, we reorder the matrix multiplication sequence, similar to the join-order optimization in databases.
② *Local result assembly.* The main task here is to detect and remove duplicate computations by applying 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 $\circ$ (element-wise multiplication) between $I_2D_2M_2^T$ and the redundancy matrix R_2. This way, we drop the redundant intermediate results indicated by the redundancy matrix R_2. Figure 4c shows the results of T_1X and $(T_2 \circ R_2)X$. It is easy to verify that their addition is the same as TX.

DI metadata & factorization. First, to compute the local LMM result, in the above rule (1) [27], X is partitioned as $X[1:c_{S_1}]$ and $X[c_{S_1}+1:c_T,]$ 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 local LMM results (i.e., $D_1X[1:c_{D_1},]$ and $D_2X[c_{D_1}+1:c_T,]$) are simply added up via indicator matrices I_1 and I_2. However, as we have shown, we need to handle redundancy when generalizing the LA factorization problem.

Challenges. Based on the process we described, we showed a simple example of how DI metadata can be used in ML factorization. Nonetheless, such processes might be less straightforward due to more complex schema or row mappings, produced by the corresponding DI 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.

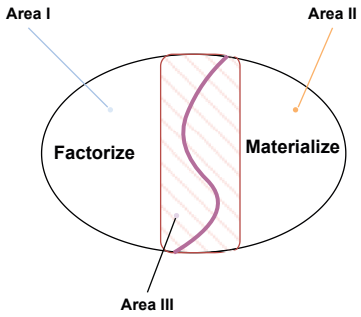

Figure 5: An abstraction of different decision areas (factorize/materialize) and their boundaries

B. Cost estimation challenge: to factorize or to materialize

Problem analysis. Factorization has been shown to be effective at increasing the efficiency of model training [27]–[29], [50], [50]–[57]. However, the question of *when* to factorize, is not fully answered. We illustrate the problem intuitively in Figure 5. Let us assume that there exists a borderline (the curvy purple line), between the 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 [27] 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; it is known which LA operators are affected by factorization and which are not [27]. To examine the relative speedup of factorization, we mainly need to inspect the interactions between physical data transfers (e.g., network and memory bandwidth) [58], and data redundancy [27], [50]. 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. In the following, we explain why data silos bring more challenges and research opportunities.

Cost estimation depends on data relationships. To differentiate area I from the rest in Figure 5, two heuristic rules are proposed based on tuple ratio and column ratio (i.e., feature ratio) between source tables and target tables [27]–[29] . However, when we come to complex data integration scenarios, there are more parameters to consider. Before materializing the target table, among silos there are parameters relevant for the redundancy, source description (e.g., number of sources, number of columns and rows in each source, null value ratio per table), source correspondences (column matching and row matching between sources), etc. It is challenging to calculate

		Redundancy in source tables	
		Yes	*No*
Redundancy in the target table	*Yes*	Morpheus: 70% Amalur: 70%	Morpheus: 70% Amalur: 70%
	No	Morpheus: 20% **Amalur: 80%**	Morpheus: 30% **Amalur: 70%**

Table III: Percentage of correct factorization decisions of Amalur vs Morpheus [27]

the borderline in Figure 5 and to design an accurate cost model taking such data integration metadata into account. From the preliminary experimental results in Table III[3], we observe that, by combining these parameters, in certain cases Amalur's cost estimation process can lead to better decisions than the state-of-the-art solution *Morpheus* [27]. In future work, we plan to cover a wide range of DI scenarios with more parameter combinations and rigorously test our approach.

DI metadata as cost model parameters and pruning rules. Data integration metadata is relevant in two cases. First, as discussed above, some of DI metadata could be used as parameters of a cost model. The data integration principles could also potentially affect how these parameters should be combined. Second, data integration is a topic with mature logic-based theoretical frameworks. A natural question is: *can we use these logic rules in cost estimation?*

Example IV.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 Figure 5. In similar cases, we can make a straightforward decision on choosing materialization. A tgd is a first-order sentence, which can be easily implemented and evaluated, e.g., a datalog program. In future work, we plan to study how to utilize such logical rules for cost estimation.

The above example is one of the simplest applications of mapping formalism in the context of cost estimation for factorization. There are more types of tgds describing more complicated dataset relationships, e.g., nested tgds [59], and plain SO tgds [60]. The discussion could also be expanded to more types of metadata, e.g., expand the existing entity resolution approaches [61] and come up with other pruning rules. In short, utilizing DI metadata in factorization still needs more effort from data integration theoreticians and practitioners.

[3]Experiment setting: $c_{S_1} = 1$, $c_{S_2} = 100$; we set the values of r_{S_1} as $\{10, 50, 100, 500, ..., 1000000, 5000000\}$, and r_{S_2} as $0.2 \times r_{S_1}$, respectively. We tested ten scenarios in each of the four cases in Table III. We computed the percentage of times that the cost estimation procedures correctly predicted factorization.

Summary & Opportunities

– Data integration metadata is useful for factorization over silos, and it requires much more research.
– The more complicated dataset relationships call for new cost models to tell apart factorization from materialization.

V. Data integration and Federated Learning

A. Challenge I: Automate data transformation for FL

In federated learning, 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. It costs massive workforce or programming efforts to collect, prepare and transform data from the sources, which also involves tiresome re-engineering. With our proposed mapping and indicator matrices, the subsets of local data can be represented and embedded in the federated models, which has great potential to automate the whole process. In what follows, we explain our intuition with the vertical federated linear regression (FLR) algorithm from [35] and Example 2 in Table I. The FLR training objective is:

$$\min_{\Theta_A,\Theta_B} \sum_i \left\| \Theta_A X_A^{(i)} + \Theta_B X_B^{(i)} - Y^{(i)} \right\|^2,$$

where X_A and X_B are feature spaces of S_1 and S_2 respectively, Y is the label space of S_1, Θ_A, and Θ_B are the local FLR model parameters of S_1 and S_2. i denotes the row index of data instances in the matrix.

The performance of trained FLR models depends on the quality of X_A and X_B, which are prepared before training and fixed during training. Refining the performance of FLR models typically requires regenerating X_A and X_B. With our mapping and indicator matrices, we can integrate the generation of X_A and X_B into the FLR training as an end-to-end optimization procedure. By denoting X_A as $I_1 D_1 M_1^T$ and X_B as $I_2 D_2 M_2^T$, we rewrite the FLR objective as:

$$\min_{\substack{\Theta_A,\Theta_B;\\ I_1,M_1,I_2,M_2}} \sum_i \left\| \Theta_A (I_1 D_1 M_1^T)^{(i)} + \Theta_B (I_2 D_2 M_2^T)^{(i)} - Y^{(i)} \right\|$$

Optimization and automation opportunities. The new FLR objective can be optimized by alternatively training Θ_A, Θ_B and I_1, M_1, I_2, M_2. While Θ_A, Θ_B can be trained by following the secure federated learning procedure [35], efficiently and effectively training I_1, M_1, I_2, M_2 are challenging problems. For one thing, the conventional centralized data selection approaches cannot be directly applied to the decentralized federated training because they impose heavy communication overheads. Therefore, communication-efficient data selection solutions are required to favor a fast search for optimal I_1, M_1, I_2, M_2. For another, I_1, M_1, I_2, M_2 contain metadata of different data sources, and therefore they should be trained

in a privacy-preserving manner. These challenges open up new research directions that incorporate multiple disciplines involving data integration, federated learning, and cryptography.

B. Challenge II: privacy-preserving DI+FL pipeline

Problem setting in existing FL frameworks. In existing vertical federated learning frameworks for tabular data [62]–[65], the common assumption is that entity resolution and federated learning are two isolated tasks, and entity resolution can be seen as a preprocessing step of federated learning. Moreover, they assume that the two data sources, party A and party B, do not have overlapping feature columns, as shown in the below example:
Party A: $S_1(r, m, a)$
Party B: $S_2(r, o)$, and the schema mapping is
$m_1: \ \forall r, m, a, o \ (S_1(r, m, a) \wedge S_2(r, o) \rightarrow T(m, a, o))$

Source table S_1 of party A and source table S_2 of Party B, share a row id column r, which is not a feature. The row id matching is given to the VFL framework as inputs.

However, such a problem setting is too ideal for real-life use cases. First, the results from an entity resolution approach often need to be verified by a human being. The separation between entity resolution and federated learning means that a third party is needed. She or he is trustworthy and will not leak information. Such a third party is not always available in a privacy-preserving use case. Second, as shown in Table I, the dataset relationships can be more complicated than the above example. The columns between silos could overlap, which requires a more sophisticated protocol to exchange gradients and model weights between different parties. Thus, the natural question is: *how to seamlessly apply the machine-generated ER results to jointly train models while preserving privacy?*

New challenges. The challenges of building an end-to-end entity resolution and federated learning pipeline are three-fold. First, we need to define the representation for the machine-generated ER results, which are most likely approximate. Then the question is how to use such ER results in federated learning without affecting model accuracy significantly. Second, such a representation should not leak information about private data. Intuitively, we can understand privacy preservation as: each party will not learn new information about other parties during learning process, which they did not know before. The common techniques for privacy-preserving in federated learning and data integration include homomorphic encryption [66], [67], secret sharing [68], [69] and differential privacy [70]. In a system like Amalur where both entity resolution and federated learning are supported, we need to encrypt the data and also the data integration metadata. Third, there are new opportunities for time-wise efficiency improvement. It is well-known that encryption often brings tremendous computation overhead. Now besides the data, we have more to encrypt and decrypt, i.e., the metadata. The DI metadata is generally smaller, compared to data instances. However, it is unclear how much overhead the encryption of DI metadata will bring, which requires further study.

Summary & Opportunities

– Data integration metadata is promising for improving ML model accuracy, and automating data transformation pipelines and federated learning.
– More complicated dataset relationships bring more practical FL use cases, but also more challenges for encryption.

VI. Conclusion

In this work, we have explored the possibilities of bringing data integration and machine learning together. Towards this direction, we have proposed a novel data integration system Amalur, which supports data integration and machine learning over silos. We have inspected the promising challenges of representing data integration metadata, and utilizing it for factorized learning 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.

Acknowledgment

This work is co-funded by the European Union Horizon Programme call HORIZON-CL4-2022-DATA-01, under Grant Agreement No. 101093164 (ExtremeXP).

References

[1] J. M. Bos, G. A. Kalkman, H. Groenewoud, P. M. van den Bemt, P. A. De Smet, J. E. Nagtegaal, A. Wieringa, G. J. van der Wilt, and C. Kramers, "Prediction of clinically relevant adverse drug events in surgical patients," *PloS one*, vol. 13, no. 8, p. e0201645, 2018.
[2] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.
[3] A. Doan, A. Halevy, and Z. Ives, *Principles of data integration*. Elsevier, 2012.
[4] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *the VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.
[5] R. Fagin, L. M. Haas, M. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis, "Clio: Schema mapping creation and data exchange," in *ER*. Springer, 2009, pp. 198–236.
[6] D. G. Brizan and A. U. Tansel, "A. survey of entity resolution and record linkage methodologies," *Communications of the IIMA*, vol. 6, no. 3, p. 5, 2006.
[7] N. Makrynioti and V. Vassalos, "Declarative Data Analytics: a Survey," *TKDE*, p. 1, 2019.
[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, 2022.
[9] M. Schleich, D. Olteanu, M. Abo-Khamis, H. Q. Ngo, and X. Nguyen, "Learning Models over Relational Data: A Brief Tutorial," in *Scalable Uncertainty Management*, N. Ben Amor, B. Quost, and M. Theobald, Eds. Cham: Springer International Publishing, 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 *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1335–1349.
[12] H. Köpcke, A. Thor, and E. Rahm, "Evaluation of entity resolution approaches on real-world match problems," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 484–493, 2010.

[13] S. Das, P. S. GC, A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park, "Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services," in *Proceedings of the 2017 ACM International Conference on Management of 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 *2020 IEEE International Conference on Data Mining (ICDM)*, 2020, pp. 1322–1327.

[15] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "Activeclean: Interactive data cleaning for statistical modeling," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 948–959, 2016.

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

[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 *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 13–24.

[18] X. L. Dong and T. Rekatsinas, "Data integration and machine learning: A natural synergy," in *Proceedings of the 2018 international conference on management of data*, 2018, pp. 1645–1650.

[19] A. Y. Halevy, A. Rajaraman, and J. J. Ordille, "Data Integration: The Teenage Years," in *VLDB*. ACM Press, 2006, pp. 9–16.

[20] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*. Morgan & Claypool Publishers, 2019.

[21] M. Lenzerini, "Data integration: A theoretical perspective," in *PODS*. ACM, 2002, pp. 233–246.

[22] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W.-C. Tan, "Data Integration: After the Teenage Years," in *PODS*, 2017, pp. 101–106.

[23] R. Hai, S. Geisler, and C. Quix, "Constance: An Intelligent Data Lake System," in *SIGMOD*. ACM, 2016, pp. 2097–2100.

[24] Z. Li, R. Hai, A. Bozzon, and A. Katsifodimos, "Metadata representations for queryable ML model zoos," 2022.

[25] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[26] C. Quix, R. Hai, and I. Vatov, "Metadata Extraction and Management in Data Lakes With GEMMS," *CSIMQ*, no. 9, pp. 67–83, 2016.

[27] L. Chen, A. Kumar, J. Naughton, and J. M. Patel, "Towards linear algebra over normalized data," *PVLDB*, vol. 10, no. 11, 2017.

[28] S. Li, L. Chen, and A. Kumar, "Enabling and Optimizing Non-Linear Feature Interactions in Factorized Linear Algebra," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1571–1588.

[29] Z. Cheng, N. Koudas, Z. Zhang, and X. Yu, "Efficient Construction of Nonlinear Models over Normalized Data," Apr. 2021, pp. 1140–1151.

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

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

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

[33] M. Esmailoghli, J.-A. Quiané-Ruiz, and Z. Abedjan, "Cocoa: Correlation coefficient-aware data augmentation." in *EDBT*, 2021, pp. 331–336.

[34] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu, "To join or not to join? thinking twice about joins before feature selection," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 19–34.

[35] 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.

[36] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid, "Privacy preserving schema and data matching," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*. ACM, 2007, pp. 653–664.

[37] A. Sabne, "XLA : Compiling machine learning for peak performance," 2020, SIGMOD workshop DEEM, industry keynote.

[38] H. Vanholder, "Efficient inference with tensorrt," in *GPU Technology Conference*, vol. 1, 2016, p. 2.

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[40] D. He, S. C. Nakandala, D. Banda, R. Sen, K. Saur, K. Park, C. Curino, J. Camacho-Rodríguez, K. Karanasos, and M. Interlandi, "Query processing on tensor computation runtimes," *Proc. VLDB Endow.*, vol. 15, no. 11, pp. 2811–2825, 2022.

[41] Y.-C. Hu, Y. L. Li, and H.-W. Tseng, "TCUDB: Accelerating database with tensor processors," in *Proceedings of the 2022 International Conference on Management of Data*, 2022.

[42] M. Kim and K. S. Candan, "Tensordb: In-database tensor manipulation with tensor-relational query plans," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014, pp. 2039–2041.

[43] D. Koutsoukos, S. Nakandala, K. Karanasos, K. Saur, G. Alonso, and M. Interlandi, "Tensors: An abstraction for general data processing," *Proceedings of the VLDB Endowment*, vol. 14, no. 10, pp. 1797–1804, 2021.

[44] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[45] NVIDIA, "Nvidia tesla v100 gpu architecture," 2017. [Online]. Available: http://images.nvidia.com/content/volta-architecture/ pdf/volta-architecture-whitepaper.pdf

[46] A. R. Benson and G. Ballard, "A framework for practical parallel fast matrix multiplication," *ACM SIGPLAN Notices*, vol. 50, no. 8, pp. 42–53, 2015.

[47] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

[48] A. Deutsch, L. Popa, and V. Tannen, "Query reformulation with constraints," *ACM SIGMOD Record*, vol. 35, no. 1, pp. 65–73, 2006.

[49] R. Chirkova, J. Yang *et al.*, "Materialized views," *Foundations and Trends® in Databases*, vol. 4, no. 4, pp. 295–405, 2012.

[50] M. Schleich, D. Olteanu, and R. Ciucanu, "Learning Linear Regression Models over Factorized Joins," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 3–18.

[51] A. Kumar, J. Naughton, and J. M. Patel, "Learning generalized linear models over normalized data," in *SIGMOD*, 2015, pp. 1969–1984.

[52] A. Kumar, M. Jalal, B. Yan, J. Naughton, and J. M. Patel, "Demonstration of Santoku: optimizing machine learning over normalized data," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1864–1867, Aug. 2015.

[53] R. Alotaibi, B. Cautis, A. Deutsch, and I. Manolescu, "Hadad: A lightweight approach for optimizing hybrid complex analytics queries," in *SIGMOD*, 2021, pp. 23–35.

[54] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich, "AC/DC: In-Database Learning Thunderstruck," in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*. Houston TX USA: ACM, Jun. 2018, pp. 1–10.

[55] J. V. D'silva, F. De Moor, and B. Kemme, "AIDA: abstraction for advanced in-database analytics," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1400–1413, Jul. 2018.

[56] J. V. D'silva, F. De Moor, and B. Kemme, "Making an RDBMS data scientist friendly: Advanced in-database interactive analytics with visualization support," *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 1930–1933, 2019.

[57] M. Schleich, D. Olteanu, M. Abo Khamis, H. Q. Ngo, and X. Nguyen, "A layered aggregate engine for analytics workloads," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1642–1659.

[58] M. Zhao, N. Agarwal, A. Basant, B. Gedik, S. Pan, M. Ozdal, R. Komuravelli, J. Pan, T. Bao, H. Lu, S. Narayanan, J. Langman, K. Wilfong, H. Rastogi, C.-J. Wu, C. Kozyrakis, and P. Pol, "Understanding data storage and ingestion for large-scale deep recommendation model training," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ACM, jun 2022.

[59] P. G. Kolaitis, R. Pichler, E. Sallinger, and V. Savenkov, "On the language of nested tuple generating dependencies," *TODS*, vol. 45, no. 2, pp. 1–59, 2020.

[60] M. Arenas, J. Pérez, J. Reutter, and C. Riveros, "The language of plain SO-tgds: Composition, inversion and structural properties," *Journal of Computer and System Sciences*, vol. 79, no. 6, pp. 763–784, 2013.
[61] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis, "An overview of end-to-end entity resolution for big data," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–42, 2020.
[62] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, no. 6, pp. 87–98, 2021.
[63] F. Fu, H. Xue, Y. Cheng, Y. Tao, and B. Cui, "BlindFL: Vertical federated machine learning without peeking into your data," in *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 2022, pp. 1316–1330.
[64] F. Fu, Y. Shao, L. Yu, J. Jiang, H. Xue, Y. Tao, and B. Cui, *VF2Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning*. Association for Computing Machinery, 2021, pp. 563–576.
[65] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. Wang, L. Wang, J. Zhou, and B. Zhang, "Large-scale secure XGB for vertical federated learning," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 443–452.
[66] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *EURASIP Journal on Information Security*, vol. 2007, pp. 1–10, 2007.
[67] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.
[68] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[69] A. Beimel, "Secret-sharing schemes: A survey," in *International conference on coding and cryptology*. Springer, 2011, pp. 11–46.
[70] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.