DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Cost Estimation for Factorized Machine Learning in Data Integration Scenarios

Author:
Jessie VAN SCHIJNDEL

Supervisors:
Dr. Rihan HAI
Dr. Asterios KATSIFODIMOS

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Web Information Systems Group
Software Technology

Student number:	5407397	
Thesis committee:	Prof. Dr. Ir. G.J. Houben,	TU Delft, chair
	Dr. R. Hai,	TU Delft, supervisor
	Dr. A. Katsifodimos,	TU Delft, supervisor
	Dr. Y. Chen	TU Delft

December 6, 2022

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Electrical Engineering, Mathematics and Computer Science
Software Technology

Master of Science

Cost Estimation for Factorized Machine Learning in Data Integration Scenarios

by Jessie VAN SCHIJNDEL

The workflow of a data science practitioner includes gathering information from different sources and applying machine learning (ML) models. Such dispersed information can be combined through a process known as Data Integration (DI), which defines relations between entities and attributes. When all information is combined in one source suited for ML purposes, this source often contains duplicate data, resulting in longer operation times. Recent work has created algebraic rewrite rules for ML such that computations can be pushed down to individual sources. This method is referred to as factorized learning. In this work, we present an implementation of Amalur, a system for automated factorized learning that is novel because of its applicability in any DI scenario. Amalur shows significant speedups for some datasets and models, while for some other datasets and models, our implementation of factorized learning results in slowdowns. Whether factorized learning is efficient depends on the operations involved in the applied ML model and the data supplied to the model. The process of estimating the efficiency of factorization is known as cost estimation. Previous efforts on cost estimation do not generalize to the DI scenarios included in Amalur, are not tailored to different ML models, and are not properly evaluated. In this thesis, we create a cost estimation procedure for Amalur suitable for any DI scenario, which adapts to individual ML models. To evaluate this procedure we create a data generator capable of generating customizable DI scenarios. Our method outperforms its competitor on DI scenarios covered by the state of the art, and shows comparable performance on newly covered DI scenarios.

Acknowledgements

I want to thank my supervisor Dr. Rihan Hai for providing me the opportunity to work on this topic, for sharing her knowledge with me and for introducing me to the art of research. Due to your supervision I have felt involved in the project and welcome as part of the Delta group. I am grateful to Dr. Asterios Katsifodimos for his insightful discussion points, ideas and advice. I thank Dr. Geert-Jan Houben for his valuable feedback during the intermediate assessments, and Dr. Lydia Chen for agreeing to be part of the committee.

Next, I want to thank the members of the Delta group for the scientific discussions and practical help. I especially want to thank Christos, Kyriakos and George, for providing me with free coffee, but more importantly, for the conversations we had over the coffee.

To my friends from back home, thank you for the lasting friendship and support. And to the students of the fourth floor, with whom I had the pleasure to work together every day, thank you for all the helpful discussions and unhelpful distractions.

Last but not least, this thesis would not have been possible without my family. I want to thank my parents Eric and Diane and my sister Renske for their support. Lastly, I express my gratitude to my boyfriend Victor, who has been with me throughout my academic career, and who has grounded me during all of it. Thank you for supporting me, encouraging me and believing in me.

Contents

List of Figures

List of Algorithms

List of Tables

List of Symbols

M	The set of Machine Learning models
X	The set of explanatory variables
X_i	The i-th explanatory variable
y	The response variable
R	The set of attribute tables
E	The entity table
R_k	The k-th attribute table
S	The set of source tables
$C/CI/SI$	The set of full / compressed / sparse indicator matrices
M / CM / SM	The set of full / compressed / sparse mapping matrices
S_k	the k-th source (entity or attribute) table
C_k / CI_k / SI_k	The full / compressed / sparse indicator matrix for S_k.
M_k / CM_k / SM_k	The full / compressed / sparse mapping matrix for S_k.
T	The target table
r_T / r_{S_k}	Number of rows in T / S_k
c_T / c_{S_k}	Number of columns in T / S_k
j_T	The join type of T
t_T	Target redundancy of T (percentage of rows in T with the same foreign key)
p_T	Sparsity of T (number of non-empty records$/(r_T \times c_T)$)
$q_r(S_k)$	The row ratio of table S_k (r_{S_k} / r_T)
$q_r(S_k)$	The column ratio of table S_k (c_{S_k} / c_T)

Chapter 1

Introduction

Nowadays, data science practitioners use large amounts of data to create Machine Learning (ML) models. Often, this data is collected from different sources, resulting in disparate datasets with varying data types and schemas. With the help of Data Integration (DI) tools [17], [19]–[21], [30], [36], relations can be discovered between tables, attributes, and entities in such datasets. Take the example shown in Figure 1.1. Here, a pharmacist has obtained data detailing the prescriptions of different patients. He wishes to use DI to combine this data with two new tables, which store patient information from two hospitals. He aims to predict the dosage of a particular type of medicine based on a patient's weight and height.

first name	last name	weight
Eric	Woodsen	60
Olivia	Burke	55
Bart	Bass	76
...	...	...

Patients hospital A

name	height	weight
Eric Woodsen	182	60
Olivia Burke	171	55
Jenny Humphrey	165	57
...	...	...

Patients hospital B

name	medicine	dosage
Eric Woodsen	5002	0.3
Eric Woodsen	2035	0.1
Olivia Burke	1004	0.005
Bart Bass	5002	0.4
Bart Bass	1004	0.01
Jenny Humphrey	1001	0.3
Jenny Humphrey	2067	0.32
...	...	...

Prescriptions

FIGURE 1.1: Example DI scenario.

Traditionally, to create input to an ML model, data scientists joined these tables. The resulting table is called a materialized table. In most cases, we want to avoid materializing joins as this leads to data redundancy. To showcase this, we perform an outer join between the three tables described in Figure 1.1. To perform this join, we compare the values stored in the columns describing the patient names. When two rows in different tables describe the same patient, their data is combined into one row. As we perform an outer join, when a row in one table does not match any rows in another table, the row is still included in the join result. The result of this join is shown in Figure 1.2. In this figure we see that more data is stored in the materialized table than in the original tables. This is because each patient may have multiple prescriptions. Data redundancy results in the utilization of extra storage and leads to runtime inefficiencies when creating ML models [5].

name	weight	height	medicine	dosage
Eric Woodsen	60	182	5002	0.3
Eric Woodsen	60	182	2035	0.1
Olivia Burke	55	171	1004	0.005
Bart Bass	76		5002	0.4
Bart Bass	76		1004	0.01
Jenny Humphrey	165	57	1001	0.3
Jenny Humphrey	165	57	2067	0.32
...		...	...	...

FIGURE 1.2: Example of a join in a DI scenario.

Recent work has created rewrites of ML models such that the materialization of joins is no longer necessary [5]–[7], [12], [13], [23]–[25], [31], [33], [34]. Instead, the computations involved in training ML models are rewritten such that they can be performed using a collection of tables and mappings describing how rows between these tables are related. When we write ML models this way, we refer to the models as factorized ML models.

To create a factorized ML model for the example in Figure 1.1, we need to know how the rows of these tables are related, but we also need to know how the columns are related. In addition, the join result shows missing data as a result of the join. Previous work has not considered such scenarios [5], [31]. In order to be able to investigate DI scenarios like the one shown in Figure 1.1, we implement factorization as described in a novel system called Amalur. Amalur aims to expand current work on factorization by supporting relations between columns as well as relations between rows and by supporting multiple types of joins.

While the factorization of ML models can boost their performance, it has been shown that this is not always the case [5], [25]. In order to choose the right approach, an effective cost estimation procedure is needed. Cost estimation procedures predict in which cases factorized ML is more efficient than performing ML on the materialized data. Prior approaches to cost estimation use simple cost models based on heuristic decision rules [5], [31]. These cost models are based on the number of rows and columns of each table and are not evaluated by the authors. As we will show in this thesis, such cost estimation approaches do not generalize to the scenarios supported by Amalur.

In this thesis, we implement Amalur, a system for factorized ML that, contrary to the state-of-the-art, can handle both row matches, column matches, and different types of joins. The cost estimation procedures by previous work [5], [24], [31] use a limited number of variables and are not evaluated by the authors. In addition, they may not generalize to the scenarios supported by Amalur. This thesis is the first to design and extensively evaluate a cost estimation procedure for factorized ML in DI scenarios.

1.1 Research questions

The research questions of this thesis are the following:

RQ1 How can we create an efficient implementation of ML factorization in Amalur?

RQ2 How can we generate data integration scenarios on which to test systems for factorized ML?

RQ3 How can we effectively perform cost estimation in Amalur?

(a) How can we predict the cost of performing ML on materialized data?

(b) How can we predict the cost of performing factorized ML?

1.2 Contributions

The contributions of this thesis can be summarized as follows:

C1 An implementation of an efficient data representation and API for ML factorization in Amalur.

C2 A data generator to generate user-specified data integration scenarios.

C3 An evaluation of Amalur on data integration scenarios generated by the data generator.

C4 The design, implementation and evaluation of a cost estimation procedure for Amalur.

1.3 Outline

The structure of this thesis is as follows. In Chapter 2 we present background information on database normalization and data integration. In Chapter 3 we outline the state-of-the-art on factorized ML. In Chapter 4 we describe our implementation of Amalur, which extends upon the current state of the art on factorized learning by making it more applicable to DI scenarios. In Chapter 5 we describe the design and implementation of our data generator. In Chapter 6 we describe the design and implementation of our cost estimation procedure for Amalur. In Chapter 7 we evaluate Amalur and our cost estimation procedure for Amalur using our data generator. Finally, in Chapter 8 we conclude by summarizing our work, answering our research questions and proposing directions for future work.

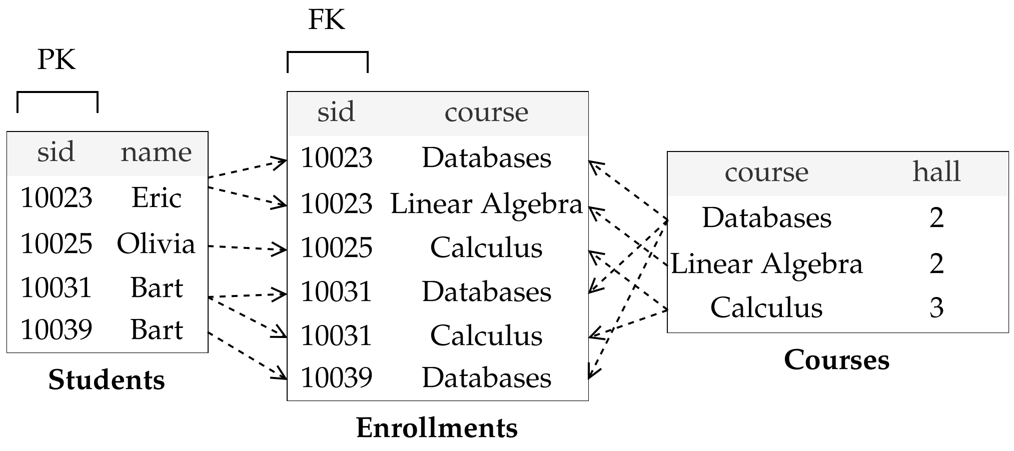
Chapter 2

Preliminaries

In this chapter, we present background information on relational databases in Section 2.1, describe the concept of database normalization in Section 2.2, distinguish different types of joins in Section 2.3 and define the concept of data integration in Section 2.4.

2.1 Relational Databases

We define a database D as a set of K tables $\mathbf{S} = \{S_1, ..., S_K\}$, where each table S_k consists of r_{S_k} rows and c_{S_k} columns. In a **relational database**, tables can be related to other tables through shared columns [10]. An example of a relational database is shown in Figure 2.1.



sid	name
10023	Eric
10025	Olivia
10031	Bart
10039	Bart

sid	course
10023	Databases
10023	Linear Algebra
10025	Calculus
10031	Databases
10031	Calculus
10039	Databases

course	hall
Databases	2
Linear Algebra	2
Calculus	3

FIGURE 2.1: A set of tables in a relational database.

2.1.1 Types of keys

In relational databases, keys are used to uniquely identify rows in tables. There are different types of keys. If a set of attributes X in table S has no two identical rows, then X is a **superkey** of S. If there are no subsets of the superkey X which are also superkeys, then we consider X a **key**. If a table has more than one key, each of these keys is called a **candidate key**. A **non-prime** attribute is not part of any candidate key of the relation.

If there is only one key in the table, this key is called the **primary key** (PK). A **foreign key** (FK) in table S links to a primary key in a different table in the database. The table containing the foreign key is referred to as the **child table**, while the table containing the primary key is referred to as the **parent table**. In Figure 2.1 we see an example of a set of tables in a relational database. Here, parent table *Enrollments* is connected to child table *Students* through its Foreign Key *sid*, which refers to the primary key *sid* in the child table.

2.1.2 Relational schema

A **relational schema** describes the tables in a relational database, the columns of each table, relations between tables and the primary and foreign keys. PKs are highlighted by underlining them. The relational schema of the example shown in Figure 2.1 is the following:

Students(<u>sid</u>, name), Enrollments(sid, course), Courses(course, hall)

There are different types of relational schemas. The most well-known schema is the **star schema**. In this schema, a single table E stores entity data, while a set of smaller tables $\mathbf{R}$ stores attribute data. The example in Figure 2.1 is a star schema where E is the table *Enrollments* and tables *Students* and *Courses* together form $\mathbf{R}$.

2.1.3 Types of relationships

We can categorize the types of relationships between entities into three relationship types: one-to-one (1:1), one-to-many (1:N) and many-to-many (M:N). In a **one-to-one relationship**, each entity A is associated with exactly one or zero instances of entity B and vice versa. For example, if we were to create a table with persons and a table with fingerprints, each person would be associated with one or zero fingerprints, and vice versa. In a **one-to-many relationship**, one instance of entity A is associated with zero or more instances of entity B, but each instance of entity B is associated with exactly one or zero instances of entity A. For example, in Figure 2.1, each course is associated with exactly one lecture hall, but each lecture hall can be associated with multiple courses. In a **many-to-many relationship**, each entity A is associated with zero or more instances of entity B and vice versa. In Figure 2.1, each student can be enrolled in multiple courses, and each course can have enrollments from multiple students.

2.2 Database Normalization

One of the strengths of relational databases is that they can be normalized. A normalized database is defined by how attributes are divided over tables and the relationships between attributes. Normalization reduces duplicate data and encourages data integrity. In order for databases to be in the most optimal normalized representation, requirements for Normal Forms have been described [9]. Different levels of Normal Forms describe stricter requirements to reduce redundancy and increase integrity.

Normal Forms are described through a concept known as functional dependency. A **functional dependency (FD)** is a constraint between two sets of attributes in a table S. Let's take X and Y as our sets of columns, then $X, Y \subseteq S$. We say that X functionally determines Y ($X \rightarrow Y$) if and only if for all rows r_i, r_j in table S, if $r_i[X] = r_j[X]$, then $r_i[Y] = r_j[Y]$. An FD is non-trivial if $Y \notin X$. An example of a dataset with FDs is shown in Figure 2.2. In this table, we have the non-trivial FD Student ID $\rightarrow$ Student name.

sid	name	course
10023	Eric	Databases, Linear Algebra
10025	Olivia	Calculus
10031	Bart	Databases, Calculus, Databases
10039	Bart	Databases

Students

FIGURE 2.2: A table with a non-trivial FD.

First Normal Form In the first Normal Form (1NF) no attribute can be multi-valued. In Figure 2.2, column *course* contains multiple values in some cells. Therefore, it is not in 1NF. Figure 2.3 shows the table in Figure 2.2 in 1NF.

sid	name	course
10023	Eric	Databases
10023	Eric	Linear Algebra
10025	Olivia	Calculus
10031	Bart	Databases
10031	Bart	Calculus
10039	Bart	Databases

Students

FIGURE 2.3: A table in 1NF.

Second Normal Form A table S in the second normal form (2NF) has to be in 1NF. In addition, in a table in 2NF, every non-prime attribute X of S is fully functionally dependent on the primary key of table S. Full functional dependency of attribute Y on X holds when Y is functionally dependent on X and not functionally dependent on any subset of X. Consider the table in Figure 2.3 again. The set of attributes [*sid, course*] is both a superkey, a key and the only key. Therefore this set of attributes is also the PK. Attribute *name* is a non-prime attribute, but it is not fully functionally dependent on the PK. Therefore, we have to make adjustments, resulting in Table 2.4.

sid	name
10023	Eric
10025	Olivia
10031	Bart
10039	Bart

Students

sid	course
10023	Databases
10023	Linear Algebra
10025	Calculus
10031	Databases
10031	Calculus
10039	Databases

Enrollments

FIGURE 2.4: A table in 2NF.

Third Normal Form To continue with our example, we extend the tables in Figure 2.4 into the tables in Figure 2.5.

sid	name
10023	Eric
10025	Olivia
10031	Bart
10039	Bart

Students

sid	course	hall
10023	Databases	2
10023	Linear Algebra	2
10025	Calculus	3
10031	Databases	2
10031	Calculus	3
10039	Databases	2

Enrollments

FIGURE 2.5: A second table in 2NF.

A table S is in the Third Normal Form (3NF) if, whenever we have a non-trivial FD $X \rightarrow Y$, either X is a superkey or Y is a prime attribute. In Table *Enrollments* in Figure 2.5, we have the key [*sid, course*]. However, we also have the FD Course name $\rightarrow$ Lecture hall, while *course* is not a superkey and Lecture hall is not a prime attribute. Therefore, Table *Enrollments* does not satisfy 3NF. Transforming this table into 3NF results in the set of tables in Figure 2.6.

sid	name
10023	Eric
10025	Olivia
10031	Bart
10039	Bart

Students

sid	course
10023	Databases
10023	Linear Algebra
10025	Calculus
10031	Databases
10031	Calculus
10039	Databases

Enrollments

course	hall
Databases	2
Linear Algebra	2
Calculus	3

Courses

FIGURE 2.6: A table in 3NF.

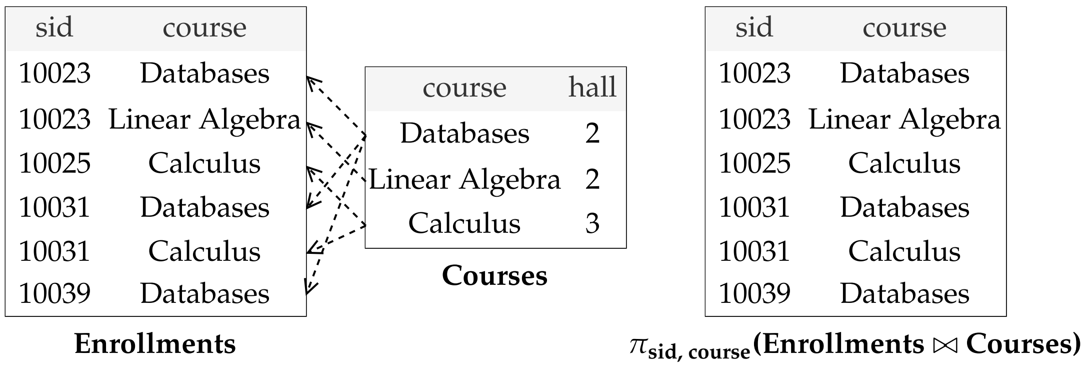
2.3 Joins

A common task in relational databases is the joining of multiple tables. Another term used to describe performing a join is materialization. A join between tables combines their data into one table, which we refer to as the materialized table. Depending on the join condition, the type of join and the structure of the data, the resulting materialized table has more or less data than the original tables. In the follow sections, we explain varying join conditions and join types.

2.3.1 Join conditions

A join condition specifies which columns we will compare and how we compare these columns. In the next sections, we describe equi joins and non-equi joins.

Equi join In an equi join, two tables are joined based on the equality of the values in one or more columns in each table. An example of an equi join is shown in Figure 2.7. Here, we join tables *Enrollments* and *Courses* from the example in Figure 2.1 based on their shared column *course*. If two column values are equal, their rows are matched. In the example, we use a projection (π) to select two columns from the join result.



sid	course
10023	Databases
10023	Linear Algebra
10025	Calculus
10031	Databases
10031	Calculus
10039	Databases

Enrollments

course	hall
Databases	2
Linear Algebra	2
Calculus	3

Courses

sid	course
10023	Databases
10023	Linear Algebra
10025	Calculus
10031	Databases
10031	Calculus
10039	Databases

$\pi_{\textbf{sid, course}}$**(Enrollments $\bowtie$ Courses)**

FIGURE 2.7: An equi join.

Non-equi join A non-equi join also joins two tables based on the column values in one or more columns in each table, but does not use the equality operator to compare values. Instead, other operators such as $<$, $>$, $\leq$ and $\geq$ are used for comparisons.

2.3.2 Types of joins

To illustrate different joins, we extend the tables shown in Figure 2.6. The extended tables are shown in Figure 2.8.

sid	name
10025	Olivia
10031	Bart
10039	Bart
10050	Ed

Students

sid	course
10023	Databases
10023	Linear Algebra
10025	Calculus
10031	Databases
10031	Calculus
10039	Databases
10062	Linear Algebra

Enrollments

course	hall
Databases	2
Linear Algebra	2
Mathematics	4

Courses

FIGURE 2.8: A collection of tables illustrating a DI scenario.

Inner join In an inner join between two tables, we keep only the rows in the left table which match at least one row in the right table, and the rows in the right table which match at least one row in the left table. An example of an inner join is shown in Figure 2.9. We will describe our joins using Relational Algebra (RA) [16]. Using RA, we describe an inner join between tables Students, Enrollments, and Courses as Students $\bowtie$ Enrollments $\bowtie$ Courses. This join may also be described as a natural join, as we are joining pairs of tables based on their common attributes.

sid	course	hall	name
10031	Databases	2	Bart
10039	Databases	2	Bart

Enrollments ⋈ Students ⋈ Courses

FIGURE 2.9: The inner join resulting from the tables in Figure 2.8.

Left outer join In a left join between two tables, we keep all of the rows of the left table and all of the rows in the right table which match with at least one row in the left table. There might be missing data for some columns that are unique to the right table. The left join between tables Students, Enrollments and Courses is shown in Figure 2.10, and can be described by RA as Enrollments ⟕ Students ⟕ Courses. Again, this is a natural join.

sid	course	name	hall
10023	Databases		2
10023	Linear Algebra		2
10025	Calculus	Olivia	
10031	Databases	Bart	2
10031	Calculus	Bart	
10039	Databases	Bart	2
10062	Linear Algebra		2

Enrollments ⟕ Students ⟕ Courses

FIGURE 2.10: The left join resulting from the tables in Figure 2.8.

Full outer join In a full outer join, we keep all of the rows of both tables. There might be missing data for columns that are unique to the left table and for columns that are unique to the right table. The full outer join between tables Students, Enrollments and Courses is shown in Figure 2.11 and can be described as the natural join Enrollments ⟗ Students ⟗ Courses.

sid	course	hall	name
10023	Databases	2	
10023	Linear Algebra	2	
10025	Calculus		Olivia
10031	Databases	2	Bart
10031	Calculus		Bart
10039	Databases	2	Bart
10062	Linear Algebra	2	

Enrollments ⟗ Students ⟗ Courses

FIGURE 2.11: The full outer join resulting from the tables in Figure 2.8.

2.4 Data Integration

As the amount and the variety of data available to data scientists for analysis have increased, data integration techniques are increasingly more important to scientists building machine learning models [15]. Data integration allows us to integrate heterogeneous data from a number of autonomous sources [14]. Figure 2.12 shows an example of a data integration system architecture. The bottom of this figure shows different data sources. While some data sources may be traditional relational databases, others may be sources such as web pages and text files. Each source is connected to a wrapper or extractor. These components of the data integration system extract data from the data sources. The top of the figure shows a component that is either a mediated schema or a warehouse. This component does not store data for data integration systems that use a mediated schema. Instead, it stores the logical schema which users can use to query data from the system. For data integration systems which use a warehouse, this component does store data.

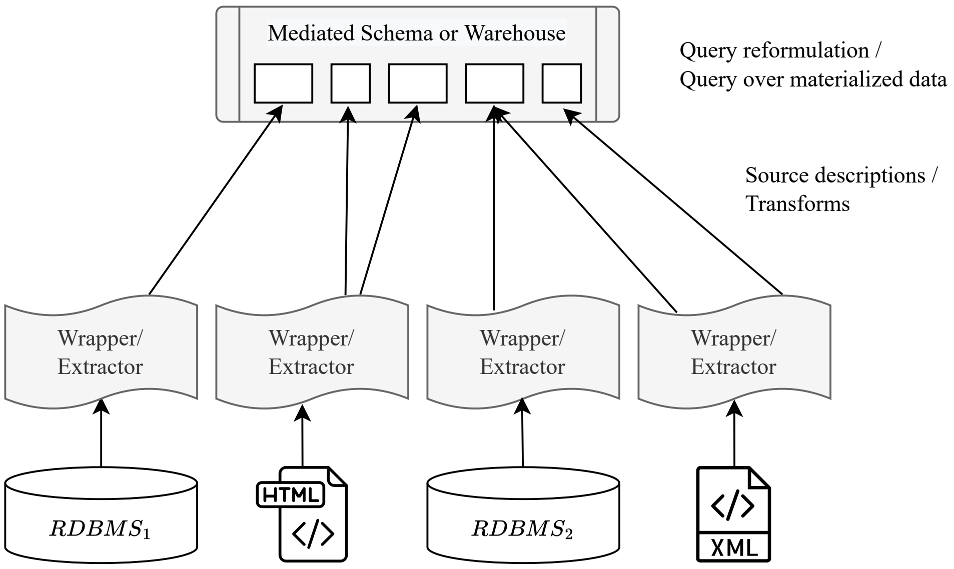


FIGURE 2.12: The basic architecture of a general-purpose data integration system. Figure created by [14].

In Chapter 1 we described an example DI scenario in Figure 1.1. Connecting this example to the architecture shown in Figure 2.12, each of the tables in Figure 1.1 could be stored in separate databases in the bottom of the architecture. The resulting mediated schema is (*name, weight, height, medicine, dosage*). This schema is obtained through two techniques known as entity matching and schema matching.

2.4.1 Entity matching

Entity matching techniques describe how entities in one table relate to entities in another table. In the case of tables in a relational database, these entities are represented as rows. For the example in Figure 1.1, entity matching aims to find which rows reference the same patient. This can be done through the column(s) describing the patient name. For the table *Patients hospital B* and the table *Prescriptions*, this match is straightforward, as both contain the column *name* describing the patient's name. For the table *Patients hospital A* however, more advanced techniques are needed, as the columns *first name* and *last name* have to be appended to discover the patient's name.

2.4.2 Schema matching and mapping

Schema matching and mapping techniques describe how columns in one table relate to columns in another table. A schema match between two columns in different tables describes how the two columns are semantically related. A schema mapping describes how we can relate the two columns. Turning again to the example in Figure 1.1, columns *name* in table *Patients hospital B* and *Prescriptions* have to be matched, and column *name* in table *Prescriptions* has to be matched to the combination of the columns *first name* and *last name* in the table *Patients hospital A*. In addition, the column *weight* has to be matched to column *weight* in *Patients hospital B*.

Chapter 3

Literature Review

This chapter outlines the state-of-the-art on factorized ML. Papers were collected through the snowball technique, starting from the paper on the Morpheus system [5] and from the paper on the Amalur system [18].

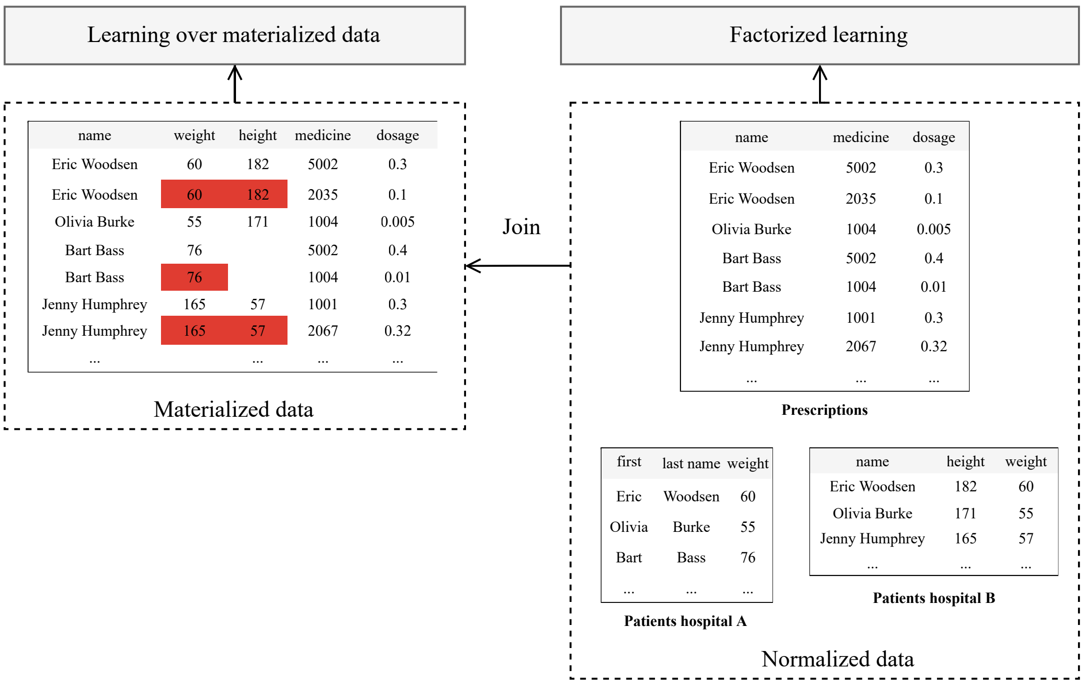
3.1 Avoiding Materialization

As discussed in Section 2.2, joins can lead to data redundancy. Therefore, researchers have investigated whether joins can be avoided before training ML models [26], [35], [37]. Avoiding a join, in this case, means not performing a join and training a model on a single table instead. [26] investigate whether joins can be skipped on data with PK-FK relationships without significantly affecting the performance of ML models. They find that for data with PK-FK relationships, we often do not need to join the entity table with the attribute tables as long as we keep the FKs in the entity table. [35] extend this notion to non-linear models and find that these models are even more resilient to avoiding joins. Another approach to reduce the cost of materialization is to return a random sample of the join, and train ML models on this sample, which has also proven to be an effective method [37].

3.2 Manual Factorization

Other works discover that it is possible to avoid joins while still including attributes from multiple tables in an ML model. This is possible through a technique known as factorized learning [25]. Factorized learning avoids data redundancy by performing computations directly on normalized data. Figure 3.1 visualizes how factorization compares to its alternative, which is learning over materialized data. Work on factorization uses the term target redundancy to explain why learning over materialized data involves more computations than factorization. To understand target redundancy, we inspect Figure 3.1. When the source tables in Figure 3.1 are joined together, one row in a table describing patients can be matched with multiple rows in the prescriptions table. This leads to duplicate or redundant data in the join result. This occurrence of duplicate data is what we describe as target redundancy.

Table 3.1 shows systems that perform ML on multi-table datasets using factorization. The authors of **Orion** [25] coined the term factorized learning. Their evaluation shows that factorized learning is often more efficient than learning over materialized data. The same authors develop **Santoku** [24], a system in which an estimate is provided of when we should apply factorized learning and when we should perform learning over materialized data. Following this line of research, a system called **F** by [33] also performs factorized learning. Where previous work was limited to inner



name	weight	height	medicine	dosage
Eric Woodsen	60	182	5002	0.3
Eric Woodsen	60	182	2035	0.1
Olivia Burke	55	171	1004	0.005
Bart Bass	76		5002	0.4
Bart Bass	76		1004	0.01
Jenny Humphrey	165	57	1001	0.3
Jenny Humphrey	165	57	2067	0.32
...		...	...	...

name	medicine	dosage
Eric Woodsen	5002	0.3
Eric Woodsen	2035	0.1
Olivia Burke	1004	0.005
Bart Bass	5002	0.4
Bart Bass	1004	0.01
Jenny Humphrey	1001	0.3
Jenny Humphrey	2067	0.32
...	...	...

Prescriptions

first	last name	weight
Eric	Woodsen	60
Olivia	Burke	55
Bart	Bass	76
...	...	...

Patients hospital A

name	height	weight
Eric Woodsen	182	60
Olivia Burke	171	55
Jenny Humphrey	165	57
...	...	...

Patients hospital B

FIGURE 3.1: Factorized learning and learning over materialized data.

joins on data with PK-FK relations, F applies to any natural join. All systems mentioned so far require continuous features or one-hot encoded categorical features. The authors of F later published a system called **AC/DC** [23]. This system also performs factorized learning, but it does not require categorical variables to be one-hot encoded. Later, the authors of AC/DC developed a system called **LMFAO** [34]. LMFAO incorporates different ML model algorithms than AC/DC and outperforms its predecessor.

3.3 Automated Factorization

Later, another paper is published by the authors of Orion [25] and Santoku [24], demonstrating a system called **Morpheus** [5]. Morpheus is a generalization of Orion. While the previous systems were created through manual factorization, where individual ML models are rewritten into factorized versions, Morpheus provides a framework of rewrite rules that can automate such rewrites. To explain how Morpheus works, we need some terminology. We refer to the individual tables in the normalized data as source tables, and we refer to the materialized table as the target table. Morpheus' rewrite rules are expressed using the normalized matrix, which is a set of matrices containing the normalized data and row mappings which map the rows in each source table to a row in the join result. Each rewrite rule specifies how a linear algebra operator can be factorized.

The authors of Morpheus identify the issue that the performance of their system highly depends on the intricacies of the programming language in which it is developed. They especially encounter issues in their Python implementation. Motivated by this, the authors of Morpheus create the system called Trinity [22], which allows factorized ML to be used in many programming languages. A system called

HADAD [1] creates rewrites for systems like Morpheus to speed up their computations. HADAD realizes such improvements through the reordering of multiplications and by exploiting pre-computed results.

Non-linear model factorization Morpheus is extended to non-linear ML models in **MorpheusFI** [31]. In order to cover these models, the authors create a new data abstraction called the interacted normalized matrix. They augment Morpheus' collection of linear algebra operators with three non-linear operators that capture quadratic interactions. The same authors later extend this work by generalizing it to a larger number of non-linear models [7]. Considering work on manual factorization, the previously mentioned system **AC/DC** [23] is a generalization of the system F to non-linear models.

ML factorization in DI scenarios Whereas most previous work focuses on extending ML factorization to new ML models, a new system called **Amalur** [18] intends to incorporate ML factorization into a DI system. DI can involve both column and row matching, so the data representation used for factorization has to be adjusted accordingly. The authors of Amalur [18] create a mapping and indicator matrix for this purpose. The indicator matrix indicates row matches, while the mapping matrix indicates column matches. Rewrite rules are adapted to this data representation. Because of the inclusion of the mapping matrix, Amalur is the first system for factorized ML which can handle the example shown in Figure 3.1.

3.4 Covered Types of Joins

While Orion [25], Santoku [24], Morpheus [5], MorpheusFI [31] and [7] work for PK-FK joins, the systems F [33] and Amalur [18] work for arbitrary join queries. The system F [33] works for joins and union queries, but is created through manual factorization instead of automated factorization. Morpheus [5] considers M:N joins in addition to PK-FK joins in a star schema.

System	Reference	Year	Join type
Orion	[25]	2015	PK-FK join
Santoku	[24]	2015	PK-FK join
F	[33]	2016	Natural join
Morpheus	[5]	2017	PK-FK join and M:N join
AC/DC	[23]	2018	Natural join
MorpheusFI	[31]	2019	PK-FK join
LMFAO	[34]	2019	Natural join
	[7]	2021	PK-FK join
HADAD	[1]	2021	Not applicable
Trinity	[22]	2021	PK-FK join
Amalur	[1]	2021	Arbitrary joins and union

TABLE 3.1: Previous work on factorized learning.

3.5 Covered Factorized Machine Learning Models

Linear models Over time, the ML models covered by ML factorization research have expanded. These models are shown in Table 3.2. The authors of Orion [25] initially focus on generalized linear models and solve these using batch gradient descent. The authors of Orion apply factorized learning to a wider range of ML models in the system Santoku [24]. The most substantial shift in the approach to factorized ML happens in Morpheus, which implements automated factorized learning. [5] Any ML model consisting of linear algebra operators can be rewritten using the rewrite rules provided in this work. The authors used their rewrite rules to create factorized algorithms for logistic regression, linear regression, K-Means, and Gaussian NMF.

Non-linear models The system F [33] performs factorization for least squares regression models. Although F does not include the wide range of ML models covered by the research which predates it [24], it allows for feature interactions in linear regression, making it the first system to include non-linear ML models. The system AC/DC [23] is the next system with non-linear models, as it includes principal component analysis and factorization machines. AC/DC expands upon the linear regression model in F by including a polynomial regression model. The system LMFAO [34] builds upon AC/DC. LMFAO shows a new approach to linear regression and decision trees and includes an approach to learning the structure of bayesian networks. In MorpheusFI [31], Morpheus is expanded to non-linear ML algorithms by allowing for feature interactions. Later, [7] extended ML factorization to gaussian mixture models and neural networks.

Type	ML model	Reference
Linear	Linear regression	[5], [22]–[25], [33], [34]
	Logistic regression	[5], [22], [24], [25], [31]
	Decision trees	[24], [34]
	Feature ranking	[24]
	Naive Bayes	[24]
	K-Means	[5], [22]
	Gaussian Non-negative Matrix Factorization	[5], [22]
	Linear Support Vector Machines	[31]
	Principal Component Analysis	[23]
	Factorization Machines	[23]
Non-linear	Gaussian Mixture Models	[7]
	Neural Networks	[7]
	Polynomial regression	[23]
	Linear regression with feature interactions	[33]
	Linear Support Vector Machines with feature interactions	[31]

TABLE 3.2: Machine Learning models factorized by previous research.

3.6 Covered Factorized Linear Algebra Operators

The operators covered by the state-of-the-art on automated ML factorization are shown in Table 3.3. For the Morpheus system, basic linear algebra operators were factorized [5]. Later, [6] extend Morpheus with factorized quadratic interactions. The authors define three non-linear operators: self-interaction, interaction, and cross-interaction. Any ML model which consists of the operators covered by a system can be rewritten to a factorized version.

	operator type	operator	Factorizable	Reference
Linear	Element-wise scalar operator	Arithmetic operator	Yes	[5], [6]
		Transpose	Yes	[5]
		Scalar function f	Yes	[5]
	Aggregation	Row summation	Yes	[5], [6]
		Column summation	Yes	[5], [6]
		Summation	Yes	[5], [6]
	Multiplication	Left multiplication	Yes	[5], [6]
		Right multiplication	Yes	[5], [6]
		Cross-product	Yes	[5]
	Inversion	Pseudoinverse	Yes	[5]
	Element-wise matrix operator	Arithmetic operator	No	[5]
Non-linear	Quadratic interaction	Self-interaction	Yes	[6]
		Interaction	Yes	[6]
		Cross-interaction	Yes	[6]

TABLE 3.3: Operators covered by the state-of-the-art on automated factorized learning. Table inspired by [5].

3.7 Cost Estimation

All works mentioned in Table 3.1 identified that, overall, factorized ML yields speedups. However, this is not always the case. [5], [25] identify a need for a method to estimate when factorized learning is not the optimal solution, which will be referred to as cost estimation. The performance of factorization depends upon the amount of redundancy present in the target table [5], known as target redundancy, which we explained in Section 3.2. Cost estimation methods for factorization aim to estimate target redundancy.

In Morpheus [5], a conservative cost estimation procedure is implemented using heuristics. Specifically, a heuristic decision rule is used based on the tuple ratio and the feature ratio. More sophisticated procedures are left for future work.

3.7.1 Tuple Ratio and Feature Ratio

The cost estimation defined in Morpheus is based on two elements: the tuple ratio and the feature ratio. This section will explain what these terms mean and why the authors chose these elements for cost estimation. In Morpheus, only PK-FK joins are considered. It is assumed that the set of tables used for training the ML model consists of one entity table E containing one or more FKs, which each reference a PK in another table in a set of attribute tables $\mathbf{R}$. The PK-FK relation implies that each FK is associated with at least one PK, and each PK is associated with at least one FK. If the number of records in the tables in $\mathbf{R}$ is small, individual rows from these tables match with many rows in table E, resulting in more target redundancy. This motivates the usage of the term feature ratio.

The feature ratio (FR) is the number of rows in table E divided by the total number of rows in the tables in $\mathbf{R}$. Similarly, the tuple ratio (TR) is defined as the total number of columns in the tables in $\mathbf{R}$ divided by the number of columns in table E. When the number of columns in the tables in $\mathbf{R}$ is larger, and when the feature ratio exceeds one, this results in more target redundancy. Based on their experiments, the authors of Morpheus define a conservative heuristic decision rule using the tuple ratio and feature ratio. When the tuple ratio is < 5 and $FR < 1$, they decide not to use factorization. In their paper on the implementation of Morpheus in Python [27], the authors highlight the importance of zeroes in tables. We will refer to the number of zeroes in a table as the sparsity of a table. The authors of [27] conclude that the higher the sparsity of the tables, the more overhead there is compared to the number of arithmetic calculations. However, sparsity was not incorporated into a cost model.

3.7.2 Cardinality estimation

Previous work has argued that to perform accurate cost estimation, cardinality estimation is needed [5], [29]. The purpose of cardinality estimation in this context is to estimate the size of the join result. Previous work suggested that cardinality estimation is needed to create accurate cost models. We further explain why cardinality estimation is needed for cost estimation in Chapter **C4**.

3.8 Discussion

In this chapter, we described the current state of the art on ML factorization. We find that current methods for automatic factorization are effective, but do not support mappings between columns and are limited to inner joins. While a cost estimation procedure has been proposed based on the dimensions of source tables, this approach has not been evaluated and is not tailored to individual operators or ML models. In addition, we do not hypothesize this approach to generalize to the scenarios we wish to include in our system for factorization.

Chapter 4

Amalur

In this chapter, we describe our first contribution **C1**, the implementation of Amalur. In Section 4.2 we describe the general system architecture of our system. Each of the subsequent sections describes a part of this architecture. Section 4.3 describes the normalized matrix, Section 4.4 describes the implemented LA operations, and section 4.5 describes the implemented ML models.

4.1 Scope

Our implementation of Amalur can handle both normalized and unnormalized data. In other words, amalur can handle any mapping between rows and columns. This includes overlapping columns and any combination of overlapping rows between tables, e.g., an inner join, a left outer join, a right outer join, a full outer join, and even unions. We assume all data is numerical. This requires special attention for null values and categorical variables. Null variables can be replaced by zeroes, and categorical variables can be transformed to one-hot-encodings. Binary variables can be encoded into zeroes and ones, where encoding the most common category into zeroes is preferred. In addition, we assume that all rows stored in the normalized data are mapped to at least one row in the target table.

4.2 System Architecture

Figure 4.1 describes the system architecture of Amalur using the example shown in Figure 3.1. The first component is the User Interface (UI). Here, the user defines a data selection from a mediated schema and selects an ML model. If the model is unsupervised, the data selection consists only of independent variables X, while if the model is supervised, the user selects both independent variables X and dependent variable y. In our example, the user selects linear regression as ML model. Then, they select columns *weight*, *height*, and *medicine* as dependent variables and column *dosage* as independent variable. The implementation of the UI is outside of the scope of this thesis. Based on the user's data selection, the mediated schema returns a normalized matrix. The normalized matrix consists of a set of source tables $\mathbf{S}$, a set of indicator matrices $\mathbf{I}$ and a set of mapping matrices $\mathbf{M}$. The details of these matrices will be explained in Section 4.3. Then, the cost estimation procedure uses the normalized matrix and the ML model to determine whether factorization should be performed or not. If the cost estimation procedure predicts that factorizated learning is not faster than learning over materialized data, the normalized data will be materialized, and the ML model will be executed on the materialized data through the NumPy API.

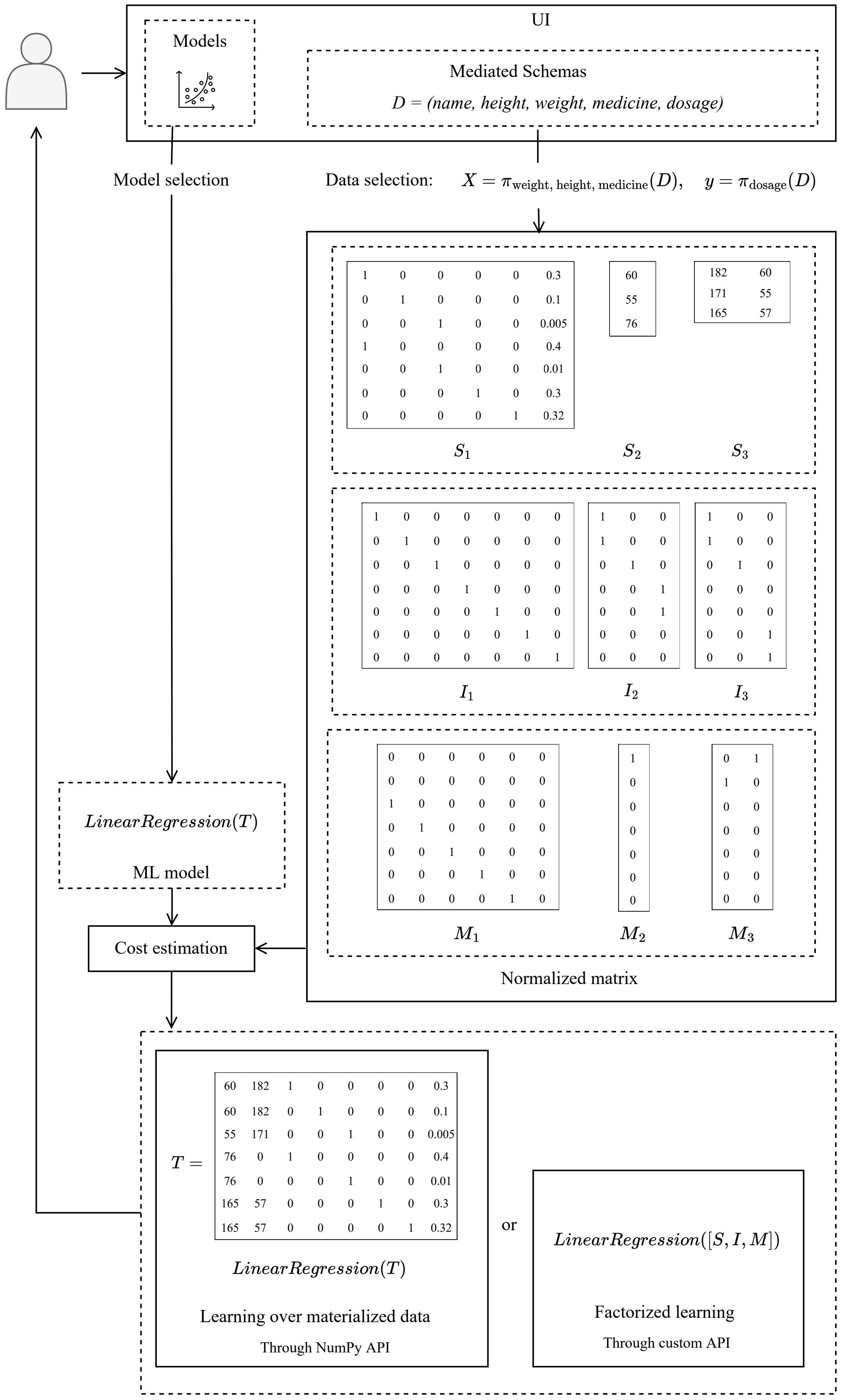


FIGURE 4.1: The system architecture of Amalur.

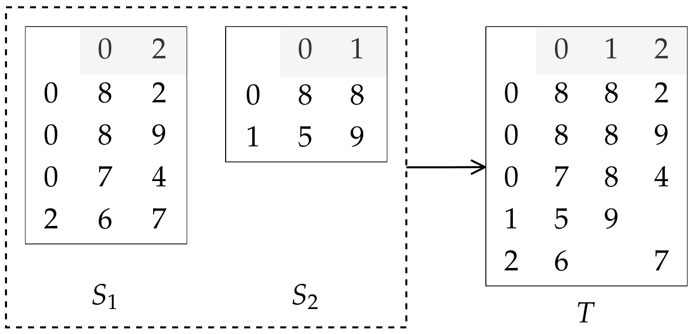
If the cost estimation procedure predicts that factorization is the better alternative, the ML model will be executed through Amalur's custom API for factorized learning. This API is implemented in this work using the normalized matrix instead of the materialized data. In the remainder of this chapter, we describe the normalized matrix, factorized LA operations and factorized ML models. In Chapter 6 we describe our cost estimation component.

4.3 Normalized Matrix

The normalized matrix consists of a set of source tables **S**, a set of indicator matrices **I** relating source table rows to target table rows, and a set of mapping matrices **M** relating source table columns to target table columns. In this section, we do not continue with the running example shown in Figure 4.1. For simplicity purposes, we create a new running example.

4.3.1 Source tables

Our set of source tables **S** contains K source tables, which are denoted $S_1, ..., S_K$. Source tables can have any number of rows and columns and can be joined through an arbitrary join or union. We use the source tables shown in Figure 4.2 as an example. The result of this join is called the target table T. We store our source tables as sparse matrices using the scipy[1] library. Specifically, we store these matrices in CSR format.


S_1

	0	2
0	8	2
0	8	9
0	7	4
2	6	7

S_2

	0	1
0	8	8
1	5	9

T

	0	1	2
0	8	8	2
0	8	8	9
0	7	8	4
1	5	9	
2	6		7

FIGURE 4.2: Two source tables and the target table which results from their join.

4.3.2 Indicator and mapping matrices

Each source table S_k has an indicator matrix I_k and a mapping matrix M_k, which describe how the rows and columns of S_k map to T.

Mapping matrices

The set of mapping matrices is denoted **M**. A mapping matrix M_k of size $c_T \times c_{S_k}$ describes how the columns of source table S_k map to the columns of target table T. An example is shown in Figure 4.4a. The mapping is created using the following definition:

[1]SciPy: `https://scipy.org/`

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

Work on the Amalur system [18] describes a compressed version of the mapping matrix. This compressed version is shown in Figure 4.4c. The compressed mapping matrix is created according to the following definition:

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

After implementing the original and compressed mapping matrices and being confronted with poor performance, we experimented with alternatives. We find that LA operations in Python are optimized for certain data structures and not for others. One library which allows users to use optimized data structures for LA is SciPy, which we also use to store our source tables. We realize improved performance when transforming the original mapping matrices and indicator matrices to sparse matrices implemented in the SciPy CSR format. This effect is shown in Figure 4.3. We achieve the best performance when the source tables, indicator matrices and mapping matrices are implemented using The SciPy CSR format. We refer to this as a sparse implementation.

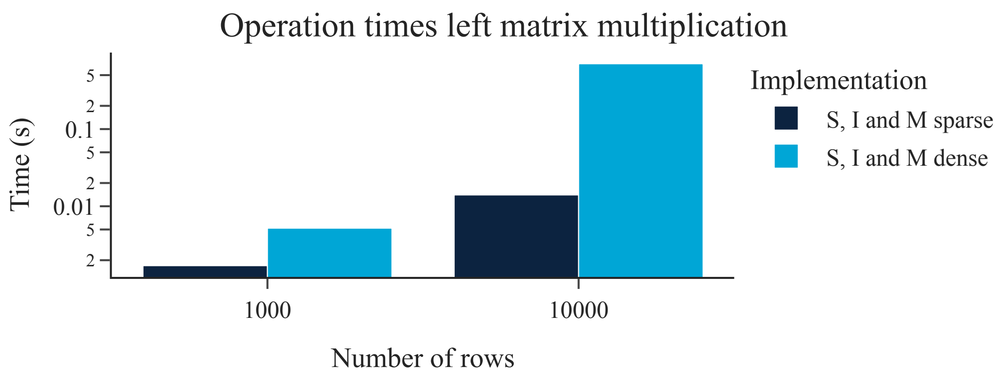


FIGURE 4.3: Operation times of left matrix multiplication for a sparse and a dense implementation. Datasets were generated using the generator described in Chapter **C2**. The number of rows in T is varied, the number of columns in T, $c_T = 100$, the number of columns in the first source table S_1, $c_{S_1} = 90$, the number of columns in the second source table S_2, $c_{S_2} = 11$, the number of rows in S_1, $r_{S_1} = r_T$ and the number of rows in S_2, $r_{S_2} = 0.05 \cdot r_T$. The generated scenario is an inner join where each row in S_2 is matched with a row in S_1.

Figure 4.4c shows the sparse matrix representation of the mapping matrices.

Indicator matrices

An indicator matrix I_k of size $c_T \times c_{S_k}$ describes how the rows of source table S_k map to the rows of target table T. An example is shown in Figure 4.5a. These matrices are created using the following definition:

$$M_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad M_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

(A) **M**

$$CM_1 = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \quad CM_2 = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}$$

(B) **CM**

	SM_1	SM_2
row	0 2	0 1
column	0 1	0 1
data	1 1	1 1

(C) **SM**

FIGURE 4.4: Mapping matrices for the example shown in Figure 4.2.

$$I_k[i,j] = \begin{cases} 1 & \text{if the } j\text{-th row of } S_k \text{ is mapped to the } i\text{-th row of } T \\ 0 & \text{otherwise} \end{cases}$$

The compressed version of the indicator matrix is shown in Figure 4.5b. The compressed indicator matrix is generated according to the following definition:

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

Figure 4.5c shows the sparse matrix representation of the mapping matrices.

$$I_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad I_2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

(A) **I**

$$CI_1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad CI_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}$$

(B) **CI**

	SI_1	SI_2
row	0 1 2 4	0 1 2 3
column	0 1 2 3	0 0 0 1
data	1 1 1 1	1 1 1 1

(C) **SI**

FIGURE 4.5: Indicator matrices for the example shown in Figure 4.2.

4.4 Factorized Linear Algebra operations

Next, we require an implementation of the LA operations included in the ML models we wish to support. These operations should follow rewrite rules which use only **S**, **SI**, and **SM**. This section defines these rewrite rules for element-wise scalar operations, aggregation operations and multiplication operations. Some of these operations return a materialized matrix, while other operations return a normalized matrix. Operations which return a normalized matrix allow us to exploit our rewrite rule again for a subsequent linear algebra operation.

4.4.1 Element-wise scalar operations

Transpose The rewrite for transpose is different from other rewrites. Instead of computing the transpose of the normalized matrix, we add a binary flag indicating that the normalized matrix has been transposed. We do not make any other changes to our normalized matrix. When a normalized matrix is transposed, other LA operations must be adapted to maintain correctness. We describe the standard and transposed rewrite rules for each of the following operations.

Arithmetic Operations The arithmetic operations covered are $+$, $-$, $*$, / and ^, but other are easily implemented. We represent any of these operations using the symbol $\oslash$. Arithmetic operations return normalized matrices. The rewritten operations are the following.

$$T \oslash x \rightarrow [S \oslash x, I, M]$$
$$x \oslash T \rightarrow [x \oslash S, I, M]$$

Since these rewrites return a normalized matrix, which can be transposed using a flag, the result in the transposed case is written as follows.

$$T^T \oslash x \rightarrow (T \oslash x)^T$$
$$x \oslash T^T \rightarrow (x \oslash T)^T$$

Scalar function f The scalar function f can be, for instance, a log function, sin function, or exp. Scalar functions also return normalized matrices. The rewrite of the scalar function is the following.

$$f(T) \rightarrow [f(S), I, M]$$

Since this rewrite returns a normalized matrix, which can be transposed using a flag, the result in the transposed case is written as follows.

$$f(T^T) \rightarrow f(T)^T$$

4.4.2 Aggregation

Row summation The row summation operation creates a vector of size $r_T \times 1$. The rewrite of row summation is the following.

$$\text{rowSums}(T) \rightarrow I_1\text{rowSums}(S_1) + ... + I_k\text{rowSums}(S_k)$$

Performing row summation on a transposed matrix is the same as performing column summation over a non-transposed matrix. We can define the rewrite in the transposed case as follows.

$$\text{rowSums}(T^T) \rightarrow \text{colSums}(T)$$

Column summation The column summation operation creates a vector of size $1 \times c_T$. The rewrite of column summation is the following.

$$\text{colSums}(T) \rightarrow \text{colSums}(I_1)S_1M_1^T + ... + \text{colSums}(I_k)S_kM_k^T$$

Performing column summation on a transposed matrix is the same as performing row summation over a non-transposed matrix. We can define the rewrite in the transposed case as follows.

$$\text{colSums}(T^T) \rightarrow \text{rowSums}(T)$$

4.4.3 Multiplication

Left Matrix Multiplication Left multiplication is also referred to as left matrix multiplication (LMM). The result of LMM between our normalized matrix and another matrix X is a matrix of size $r_T \times c_X$. Our rewrite of LMM is as follows.

$$TX \to I_1 S_1 M_1^T X + ... + I_K S_K M_K^T X$$

Performing LMM between two matrices where the first matrix is transposed is the same as multiplying the untransposed first matrix with the second matrix transposed and transposing this result. Therefore, we can define the rewrite in the transposed case as follows.

$$T^T X \to (X^T T)^T$$

Right matrix Multiplication Right multiplication is also referred to as right matrix multiplication (RMM). The result of RMM between our normalized matrix and another matrix X is a matrix of size $r_X \times c_T$. Our rewrite of RMM is as follows.

$$XT \to XI_1 S_1 M_1^T + ... + XI_K S_K M_K^T$$

Performing RMM between two matrices where the second matrix is transposed is the same as multiplying the untransposed second matrix with the transposed first matrix and transposing this result. We can define the rewrite in the transposed case as follows.

$$XT^T \to (TX^T)^T$$

4.4.4 Example

To explain how these rewrites work, we show an example of LMM on the data described in Figures 4.2, 4.4, and 4.5. Figure 4.6 shows how we perform LMM on materialized data: we multiply the materialized table with another matrix X.

$$\underbrace{\begin{bmatrix} 8 & 8 & 2 \\ 8 & 8 & 9 \\ 7 & 8 & 4 \\ 5 & 9 & 0 \\ 6 & 0 & 7 \end{bmatrix}}_{T} \times \underbrace{\begin{bmatrix} 8 \\ 8 \\ 7 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} 142 \\ 191 \\ 148 \\ 112 \\ 97 \end{bmatrix}}_{TX}$$

FIGURE 4.6: LMM on materialized data.

Figure 4.7 shows how we perform LMM in the factorized case according to the rewrite rules. According to our rewrite rule, we multiply I_k, S_k, M_k, and X for each S_k and sum the results. In this example, we have both row matches and column matches. This means that there is duplicate data in the source tables. We refer to this occurrence as source redundancy. In order to maintain correctness in the case of source redundancy, we have to remove duplicate data.

4.4.5 Removing redundancy from source tables

To remove source redundancy during computations, we set values in the source tables to zero. If there is an overlapping value between S_i and S_j, then the value will be set to zero in S_j. As we implement the source tables using sparse matrices in which zero values are not stored, this value is removed from memory. Source redundancy should, in theory, not negatively impact the speedups we can achieve through factorization. In practice, we expect that source redundancy will introduce overhead, resulting in a decrease in performance for factorization.

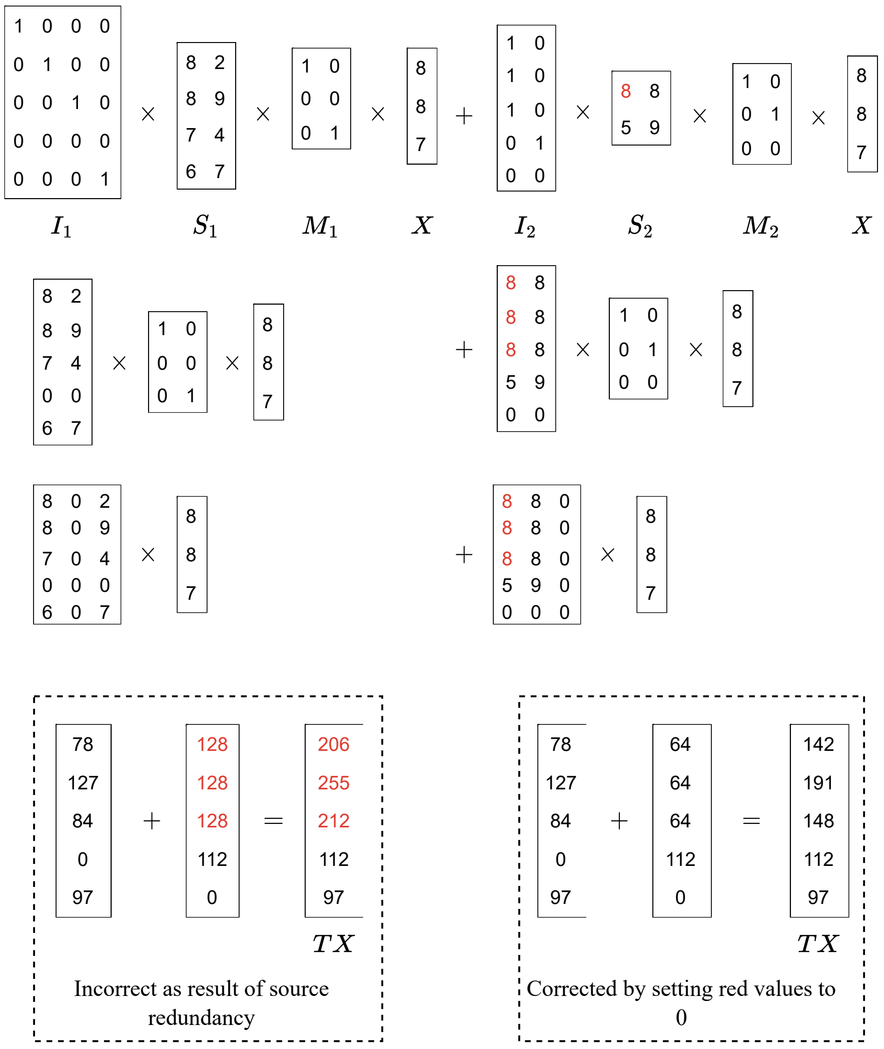

I_1
S_1
M_1
X
I_2
S_2
M_2
X
TX
Incorrect as result of source redundancy
TX
Corrected by setting red values to 0

FIGURE 4.7: Factorized LMM.

4.4.6 Materialization

Although materialization is not a linear algebra operation, we still include it in this section. Materialization is important to consider for other purposes such as testing and evaluation, and plays a role in our cost estimation. We use the definition below when we want to perform learning over materialized data.

$$T \rightarrow I_0 S_0 M_0^T + \dots + I_k S_k M_k^T$$

4.4.7 Order of operations

The problem of ordering matrix multiplications such that the total cost of multiplication is minimal is known as the matrix chain multiplication problem or the matrix chain ordering problem. The optimal ordering can be computed from the dimensions of the matrices. We calculate an optimal ordering of intermediate computations to optimize our implementation of the LA rewrites.

As an example, we inspect how to compute an optimal ordering for the multiplication $I_k \cdot S_k \cdot M_k^T$ for a source table k. Let the size of S_k be $r_{S_k} \times c_{S_k}$ and the size of T be $r_T \times c_T$. The size of I_k is $r_T \times r_{S_k}$ and the size of M_k is $c_T \times c_{S_k}$. The size of X is $c_T \times c_X$. We describe our matrix chain ordering problem as follows:

$$\underbrace{I_k}_{r_T \times r_{S_k}} \cdot \underbrace{S_k}_{r_{S_k} \times c_{S_k}} \cdot \underbrace{M_k^T}_{c_{S_k} \times c_T}$$

The number of possible parenthesizations of n matrices can be counted. We adapt the notation for this count used by Cormen [11], which is $P(n)$. Cormen defines the calculation of this number as follows:

$$P(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2. \end{cases}$$

There are two possible multiplication orderings for the three matrices, which can also be described as parenthesization options. These are $(I_k \cdot S_k) \cdot M_k^T$ and $I_k \cdot (S_k \cdot M_k^T$. The cost of the first ordering is $r_T \cdot r_{S_k} \cdot c_{S_k} + r_T \cdot c_{S_k} \cdot c_T$. This can also be calculated using the formula $r_T \cdot c_{S_k} \cdot (r_{S_k} + c_T)$. The cost of the second ordering is $r_{S_k} \cdot c_{S_k} \cdot c_T + r_T \cdot r_{S_k} \cdot c_T$., which can also be written as $r_{S_k} \cdot c_T \cdot (r_T + c_{S_k})$. We will walk through the calculations for the optimal ordering for a normalized matrix where $r_T = 10$, $c_T = 6$, $r_{S_1} = 10$, $c_{S_1} = 1$, $r_{S_2} = 1$ and $c_{S_2} = 5$. We have to perform the following two multiplication chains:

$$\underbrace{I_k}_{10\times10} \cdot \underbrace{S_k}_{10\times1} \cdot \underbrace{M_k^T}_{1\times6}$$

$$\underbrace{I_k}_{10\times1} \cdot \underbrace{S_k}_{1\times5} \cdot \underbrace{M_k^T}_{5\times6}$$

For table S_1, the first option has a lower cost, as $10 \cdot 1 \cdot (10+6) < (10 \cdot 6 \cdot (10+1)$. For table S_2, the second option has a lower cost, as $10 \cdot 5 \cdot (1+6) > 1 \cdot 6 \cdot (10+5)$.

An algorithm to find the best matrix multiplication order is included in NumPy[2], but not in SciPy. We adapt the code written for NumPy to evaluate the optimal order of operations for SciPy matrices based on the dimensions of these matrices. This implementation has two options for calculating the optimal ordering: one is only applicable for a multiplication of three matrices and one can be used to calculate the ordering for three or more matrices.The first method is described in Algorithm 1.

The second method is described in Algorithm 2, which uses the function Matrix-Chain-Order created by Cormen [11].

[2]`https://numpy.org/doc/stable/reference/generated/numpy.linalg.multi_dot.html`

Algorithm 1 Optimal matrix multiplication ordering for three matrices

```
Require: $A, B, C$
  $c_1 = r_A \cdot r_C \cdot (r_B + c_C)$
  $c_2 = r_B \cdot c_C \cdot (r_A + r_C)$
  if $c_1 < c_2$ then
      return $(A \cdot B) \cdot C$
  else
      return $A \cdot (B \cdot C)$
```

Algorithm 2 Optimal matrix multiplication ordering for any number of matrices

```
Require: $\mathbf{M}$
  procedure MATRIX-CHAIN-ORDER($\mathbf{M}$)
      $n = |M| - 1$
      Let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
      for $i \in 1 : n$ do
          $m[i,i] = 0$
      for $l \in 2 : n$ do
          for $i \in 1 : n - l + 1$ do
              $j = i + l - 1$
              $m[i,j] = \infty$
              for $k \in i : j - 1$ do
                  $q = m[i,k] + m[k+1,j] + p_{i-1}p_k p_j$
                  if $q < m[i,j]$ then
                      $m[i,j] = q$
                      $s[i,j] = k$
      return $s$
  procedure MULTI-DOT($\mathbf{M}$, $s$, $i$, $j$)
      if $i == j$ then
          return $\mathbf{M}[i]$
      return Multi-Dot($\mathbf{M}, s, i, s[i,j]$) $\cdot$ Multi-Dot($\mathbf{M}, s, s[i,j]+1, j$)
  return Multi-Dot($\mathbf{M}$, Matrix-Chain-Order($\mathbf{M}$), 0, $|M| - 1$)
```

4.4.8 Optimizations

Although the implementation described in this chapter allows us to perform factorized ML in more scenarios than previous work, we also use more computations to facilitate this. In some scenarios, we do not need these additional computations. In this subsection, we define scenarios in which we perform redundant computations. Then, we describe how we can create rewrites without these computations for these scenarios.

Scenarios without column overlap

In order to support scenarios with overlapping columns between tables, Amalur introduces the mapping matrix. However, this addition leads to more computations. For scenarios without column overlap, we do not need the mapping matrices **SM**. We adapt our rewrite rules to scenarios without column overlap by removing the multiplication of each S_k with M_k^T.

Inner and left join

While in Amalur we assume that we need an indicator matrix for each source table, previous work did not always make this assumption [5]. In Morpheus, the authors do not include an indicator matrix for the entity table in a star schema join, which is used as the first table in an inner or left join. Instead, each row in this entity table is matched to the same row in the target table. We can take this same approach in the inner and left outer join cases. We assume the entity matrix is S_1 and the corresponding indicator matrix is I_1.

Aggregation

First, we will describe how the two optimizations affect aggregation operations.

Row summation The rewrite of row summation is only affected by the second optimization. The adapted rewrite rule is the following:

$$\text{rowSums}(T) \rightarrow \text{rowSums}(S_1) + I_2\text{rowSums}(S_2) + ... + I_k\text{rowSums}(S_k)$$

Column summation The rewrite of column summation is affected by both optimizations. Therefore, we describe three cases: one case without **SM**, one case without I_1, and one case without both. First, we describe the case without **SM**. To maintain correctness, we horizontally stack the intermediate results for source tables. The corresponding rewrite rule is the following:

$$\text{colSums}(T) \rightarrow [\text{colSums}(I_1)S_1, ..., \text{colSums}(I_k)S_k]$$

For the case without I_1, the rewrite rule of column summation is the following:

$$\text{colSums}(T) \rightarrow \text{colSums}(S_1)M_1^T + \text{colSums}(I_1)S_1M_1^T + ... + \text{colSums}(I_k)S_kM_k^T$$

For the cases without **SM** and I_1, we are again horizontally stacking instead of aggregating the intermediate results. The adapted rewrite rule is the following:

$$\text{colSums}(T) \rightarrow [\text{colSums}(S_1), \text{colSums}(I_1)S_1, ..., \text{colSums}(I_k)S_k]$$

Multiplication

Next, we will describe how the two optimizations affect the multiplication operations.

Left Matrix Multiplication Like column summation, LMM is affected by both optimizations. To create the adapted rewrites, we need a new notation. We define c'_k as $\sum_{k=0}^{k-1} c_{S_k}$, which defines the number of combined columns in all source tables before source table k. For the case without **SM**, LMM does not require horizontally stacking the result. Instead, we are indexing matrix X using c'_k. The adapted rewrite of LMM is the following.

$$TX \rightarrow I_1S_1X[0 : c_{S_1}] + ... + I_KS_KX[c'_k : c'_k + c_{S_k}]$$

For the case without I_0, the rewrite of LMM is the following.

$$TX \to S_1M_1^TX + I_2S_2M_2^TX + ... + I_KS_KM_K^TX$$

For the case without **SM** and I_1, we are again indexing matrix X using c_k'. The rewrite of LMM is the following:

$$TX \to S_1X[0 : c_{S_1}] + I_2S_2X[c_{S_1} : c_{S_1} + c_{S_2}] + ... + I_KS_KX[c_k' : c_k' + c_{S_k}]$$

Right matrix Multiplication The operation RMM is also affected by both optimizations. For the case without **SM**, the adapted rewrite of RMM involves horizontal stacking and is described below.

$$XT \to [XI_1S_1, ..., XI_KS_K]$$

For the case without I_0, the adapted rewrite of RMM is the following.

$$XT \to XS_1M_1^T + ... + XS_KM_K^T$$

For the case without **SM** and without I_1, the rewrite of RMM again involves horizontal stacking and is described as follows.

$$XT \to [XS_1, ..., XI_KS_K]$$

4.5 Machine Learning Models

An ML task is defined by a response variable y, a set of explanatory variables X, and an ML model m. The ML models included in our system are linear regression, logistic regression, and Gaussian NMF. Our linear regression and logistic regression algorithms use gradient descent. We denote the set of ML models included in Amalur M. All ML models are implemented using only linear algebra operations. The relevant LA operations for each ML model in M are collected in Table 4.1.

operation	Linear regression	Logistic regression	Gaussian NMF	K-Means
T^x				✓
$T \cdot x$				✓
T^T	✓	✓		✓
rowSums(T)				✓
colSums(T)				
TX	✓	✓	✓	✓
XT			✓	

TABLE 4.1: Linear algebra operations involved in each ML model.

Linear regression Linear regression is an ML model which uses supervised learning. Linear regression aims to predict a dependent variable based on one or more independent variables using a linear relationship. The linear regression algorithm used in our system is described in algorithm 3. This algorithm uses gradient descent to compute the optimal weights.

Algorithm 3 Linear regression using Gradient Descent (from [5])

```
Require: $X, y, w, \gamma$
  for $i \in 1 : n$ do
    $w = w - \gamma(T^T((Tw) - Y))$
```

Logistic regression Logistic regression is another ML model based on supervised learning. It aims to predict the probability of a binary dependent variable based on one or more independent variables using a logistic function. The logistic regression algorithm used in our system is described in algorithm 4. This algorithm also uses gradient descent.

Algorithm 4 Logistic regression using Gradient Descent (from [5])

```
Require: $X, y, w, \gamma$
  for $i \in 1 : n$ do
    $w = w - \gamma(T^T \frac{Y}{1+e^{Tw}})$
```

Gaussian Non-Negative Matrix Factorization Gaussian Non-Negative Matrix Factorization (Gaussian NMF) is a supervised ML algorithm that represents its input matrix as two smaller matrices, of which the product approximates the input. The size of the smaller matrices is determined by the hyperparameter rank *r*. It is used for purposes such as clustering, dimensionality reduction, and feature extraction. The Gaussian NMF algorithm used in our system is described in algorithm 5 and is based on the multiplicative update rule by [28].

Algorithm 5 Gaussian NMF (from [5])

```
Require: $X, W, H$
  for $i \in 1 : n$ do
    $H = H \times (\frac{W^T T}{W^T W H})$
    $W = W \times (\frac{T H^T}{W(H H^T)})$
```

K-Means K-Means is an unsupervised ML algorithm used for clustering. It groups data in a user-specified number of clusters by defining a centroid for each cluster. The number of clusters is specified through hyperparameter *k*. Then, data points are assigned to the closest cluster centroid. The K-Means algorithm used in our system is described in algorithm 6.

Algorithm 6 K-Means (from [5])

```
Require: $X, k$
  1. Initialize centroids matrix $C_{r_X \times k}$
  $\mathbf{1}_{a\times b}$ represents a matrix filled with ones with dimension $a \times b$, it is used for replicating a vector row-wise or column-wise.
  2. Pre-compute $l^2$-norm of points for distances.
  $D_T = \text{rowSums}(T^2)\mathbf{1}_{1\times k}$
  $T_2 = 2 \times T$
  for $i \in 1 : n$ do
      3. Compute pairwise squared distances; $D_{r_X \times k}$ has points on rows and centroids/clusters o columns.
      $D = D_T - T_2 C + \mathbf{1}_{r_X \times 1}\text{colSums}(C^2)$
      4. Assign each point to the nearest centroids; $A_{r_X \times k}$ is a boolean assignment matrix.
      $A = (D == \text{rowMin}(D)\mathbf{1}_{1\times k}$
      5. Compute new centroids; the denominator counts the number of points in the new clusters, while the numerator adds up assigned points per cluster.
      $C = (T^T A)/(\mathbf{1}_{d\times 1}\text{colSums}(A))$
```

4.6 Discussion

In this chapter, we described our implementation of Amalur. This includes the implementation of sparse indicator and mapping matrices. We create an implementation of LA factorization which the user can utilize through a Python API, and an implementation of factorized ML models using this API. By using mapping matrices, we support datasets with overlapping columns. The indicator matrix supports any join or union, and also supports sequences of different types joins. To further speed up factorized learning, we calculate the optimal ordering for matrix multiplications and implement optimizations for scenarios without column overlap, and for star schema joins.

Contributions discussed in this chapter

– A flexible and efficient representation of the normalized matrix which maintains correctness even when there are both row matches and column matches between a set of tables,
– The definition of rewrite rules for element-wise scalar, aggregation and multiplication operations, including the transpose and rewrite rules for materialization,
– The implementation of four ML models using the defined rewrite rules,
– An optimization of the rewrite rules through finding the optimal order of operations for each defined operation and making adaptions for data scenarios without column overlap and for inner and left join scenarios.

Chapter 5

Data Generator

This chapter details the design and implementation of the data generator. This chapter answers **RQ2** and corresponds to contribution **C2**. The data generator is used to generate user-specified data integration scenarios. It provides the user with normalized data, materialized data and mappings between normalized and materialized data. First, we describe the scope of the data generator in Section 5.1. Section 5.2 lists parameters we use to control data integration scenarios and why we chose these parameters specifically. The subsequent sections 5.3, 5.4, and 5.5 each describe how data scenarios are generated for different joins.

5.1 Scope

Each scenario generated by the data generator is structured according to a star schema. We distinguish two types of source tables: entity tables E and the set of attribute tables R. To efficiently generate data, we add keys to our source table. For simplicity purposes, we only generate one key for each of E and R_k which is used to join all tables together. Although we are essentially generating PK-FK joins, we do not store these keys in the source tables. We also do not store column indices or names. We do not assume that each foreign key maps to exactly one primary key and that each primary key maps to at least one foreign key. Therefore, the generated data can be used to simulate joins on any set of columns or keys.

5.2 Data Parameters

The data parameters describe the characteristics of the DI scenarios. The characteristics of each R_k, E, and target table T are the size of the tables, i.e. the number of rows and the number of columns. For T, the user specifies the size of the table using absolute numbers. For E and each R_k, the table size is specified relative to the size of T. This allows us to flexibly generate a large number of scenarios in our experiments later on. Another parameter is the join type. Depending on the join type, more data parameters are required. The parameters for each join type are explained in the subsequent sections.

5.3 Inner Join

First, we describe the generation of an inner join scenario involving no redundancy. By no redundancy we mean no target redundancy and no source redundancy. To illustrate, an example is shown in Figure 5.1. From the given parameters, the generator calculates the size of table E and tables R_k. The number of columns in E is calculated as $c_{R_k} \cdot c_T$. The number of rows of each R_k (r_{R_k}) equals $q_r(R_k) \cdot r_T$ and the number of columns (c_{R_k}) equals $q_c(R_k) \cdot c_T$. As we are dealing with an inner join, the list of parameters does not include the number of rows in E. This parameter can be computed as it equals the number of rows in T.

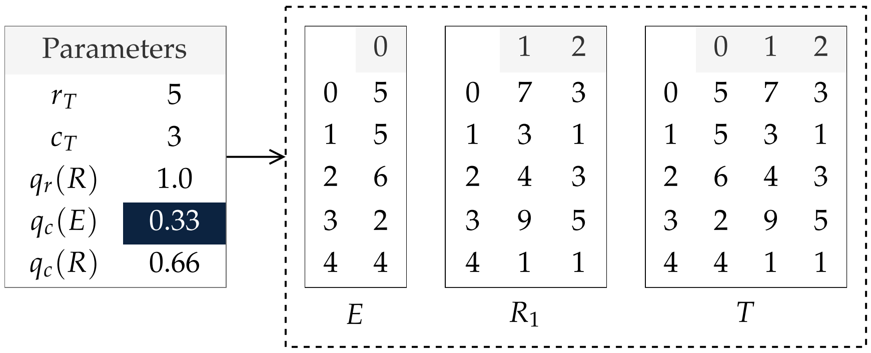


FIGURE 5.1: An inner join without redundancy.

In case the user specifies $q_r(R_k)$ is smaller than one and thus, the number of rows in R_k is smaller than the number of rows in E and T, this means that one row in R_k should be matched to multiple rows in E. In such a scenario, we are introducing target redundancy in our data. This result is shown in Figure 5.2.

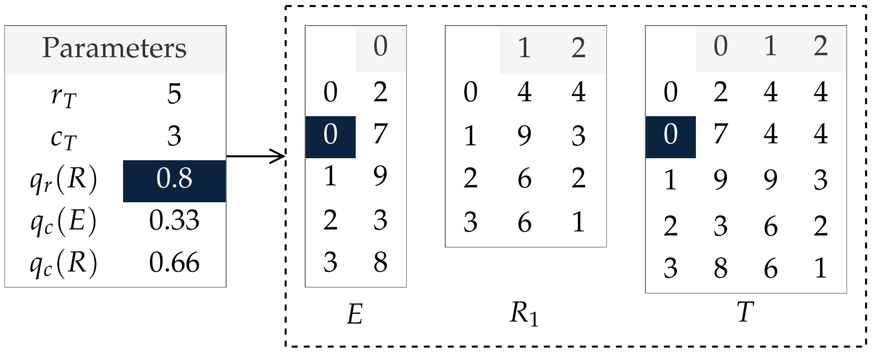


FIGURE 5.2: An inner join with target redundancy.

To create source redundancy, the user may tweak the combined number of columns in E and R_k. If the sum of all $q_r(R)$ and $q_c(S)$ is larger than one, the number of columns in E and R_k combined is larger than the number of columns in T. Therefore, some columns are shared between tables E and R_k. The result is shown in Figure 5.3.

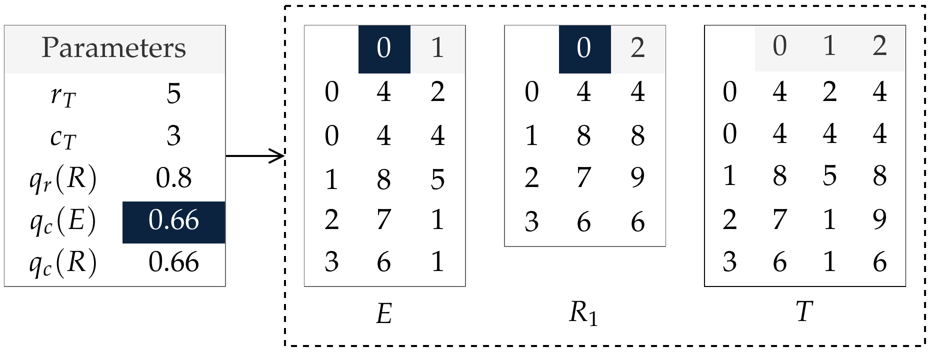
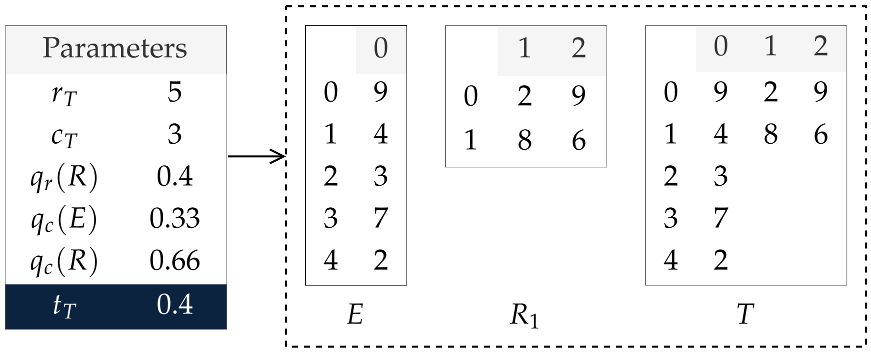


FIGURE 5.3: An inner join with target redundancy and source redundancy.

5.4 Left Join

The second type of join covered by the data generator is the left join. Figure 5.4 shows a basic example of a scenario with a left join generated by the data generator. We can no longer calculate the target redundancy from the same parameters for the left join. Therefore, we add another parameter to the list, t_T, which controls the number of rows with the same key in E. We can calculate the number of rows with target redundancy r_T as $r_T \cdot t_T$. The example in Figure 5.4 does not have target redundancy, corresponding with $t_T = 0$.



Parameters	
r_T	5
c_T	3
$q_r(R)$	0.4
$q_c(E)$	0.33
$q_c(R)$	0.66
t_T	0.4

E

	0
0	9
1	4
2	3
3	7
4	2

R_1

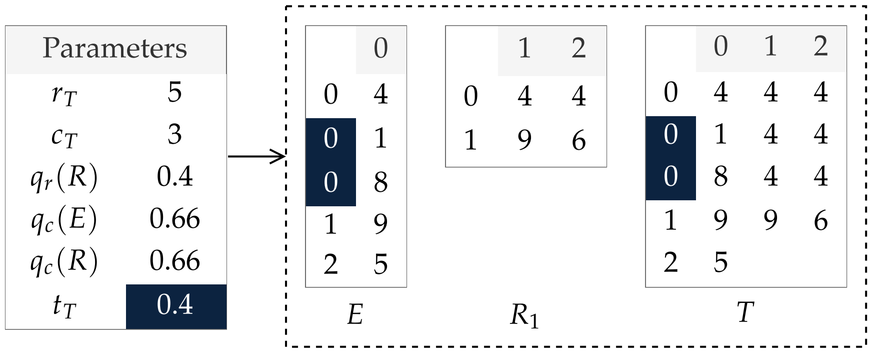
	1	2
0	2	9
1	8	6

T

	0	1	2
0	9	2	9
1	4	8	6
2	3		
3	7		
4	2		

FIGURE 5.4: A left join without redundancy.

When users want to add target redundancy, they adapt parameter t_T. This results in the example shown in Figure 5.6. However, users should be careful adjusting this parameter. The requisite $r_T + r_R - 1 < r_T$ should always hold.



Parameters	
r_T	5
c_T	3
$q_r(R)$	0.4
$q_c(E)$	0.66
$q_c(R)$	0.66
t_T	0.4

E

	0
0	4
0	1
0	8
1	9
2	5

R_1

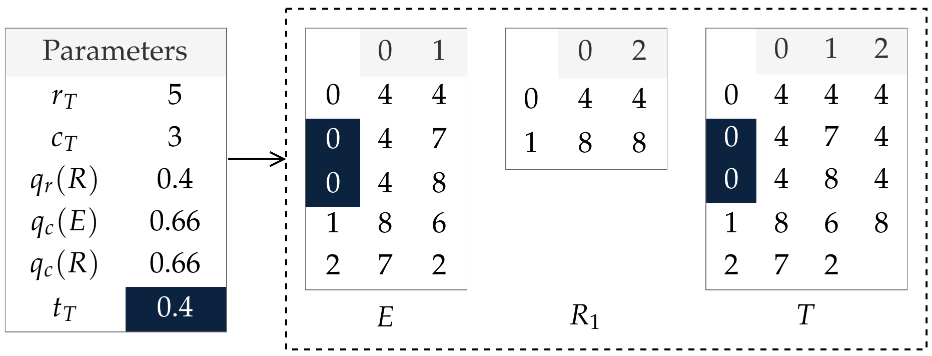
	1	2
0	4	4
1	9	6

T

	0	1	2
0	4	4	4
0	1	4	4
0	8	4	4
1	9	9	6
2	5		

FIGURE 5.5: A left join with target redundancy.

Adding source redundancy to the previous example is done in the same way as it is done for inner joins. The result is shown in Figure 5.6.



Parameters	
r_T	5
c_T	3
$q_r(R)$	0.4
$q_c(E)$	0.66
$q_c(R)$	0.66
t_T	0.4

	0	1
0	4	4
0	4	7
0	4	8
1	8	6
2	7	2

E

	0	2
0	4	4
1	8	8

R_1

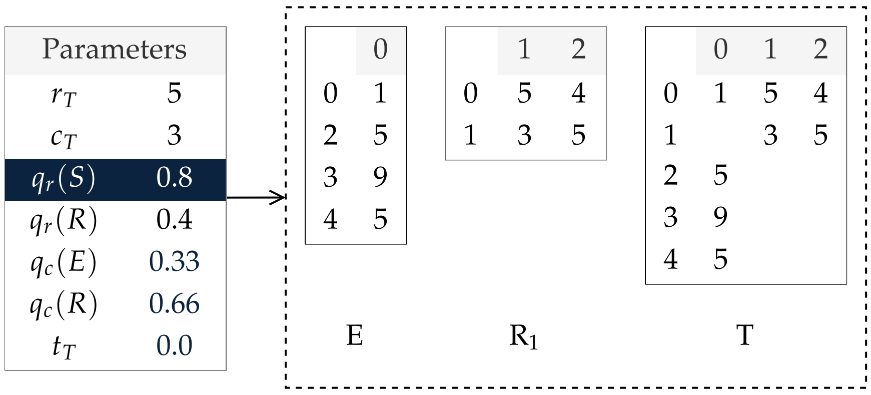
	0	1	2
0	4	4	4
0	4	7	4
0	4	8	4
1	8	6	8
2	7	2	

T

FIGURE 5.6: A left join with target redundancy and source redundancy.

5.5 Full Outer Join

To create full outer joins, another parameter needs to be added to our data generator. As shown in the example in Figure 5.7, we add the parameter $q_r(S)$. This parameter controls the number of rows in E. For the inner and left outer join, the user did need to specify this parameter, as it was equal to the number of rows in T. However, for full outer join, this does not necessarily hold. We compute the number of rows in E as $q_r(S) \cdot r_T$.



Parameters	
r_T	5
c_T	3
$q_r(S)$	0.8
$q_r(R)$	0.4
$q_c(E)$	0.33
$q_c(R)$	0.66
t_T	0.0

	0
0	1
2	5
3	9
4	5

E

	1	2
0	5	4
1	3	5

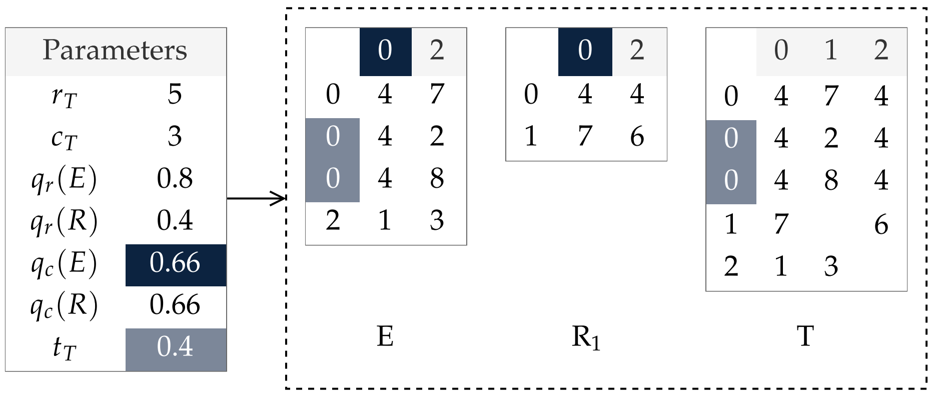
R$_1$

	0	1	2
0	1	5	4
1		3	5
2	5		
3	9		
4	5		

T

FIGURE 5.7: A full outer join without redundancy.

Again, we can add source and target redundancy. A full outer join with both types of redundancy is shown in Figure 5.8.



Parameters	
r_T	5
c_T	3
$q_r(E)$	0.8
$q_r(R)$	0.4
$q_c(E)$	0.66
$q_c(R)$	0.66
t_T	0.4

	0	2
0	4	7
0	4	2
0	4	8
2	1	3

E

	0	2
0	4	4
1	7	6

R$_1$

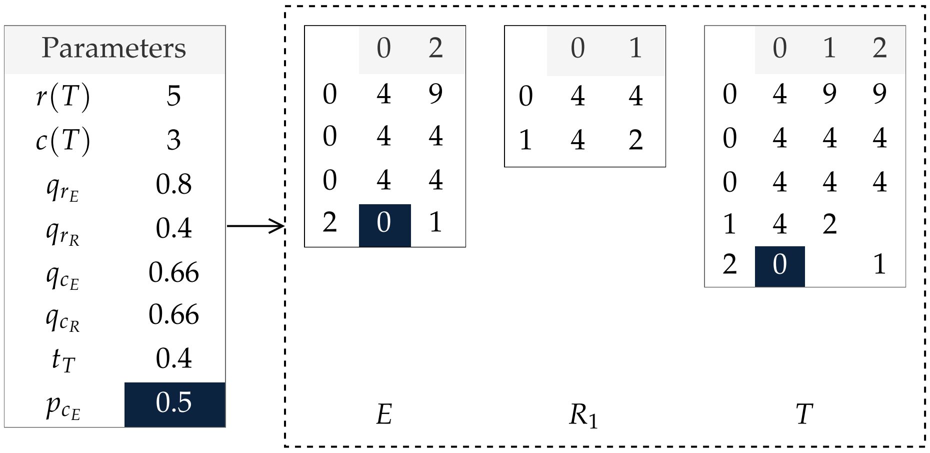
	0	1	2
0	4	7	4
0	4	2	4
0	4	8	4
1	7		6
2	1	3	

T

FIGURE 5.8: A full outer join with source and target redundancy.

5.6 Sparsity

Adopting the notation used for MorpheusPy [27], we denote sparsity using p. The sparsity p of a table S is the number of nonzero records in S, which is also denoted $nnz(S)$. In our data generator, we control parameter p_{c_E}, which denotes the percentage of columns in E whose entries consist of all zeroes. Figure 5.9 shows an example of a DI scenario with sparsity. As some data in E is repeated, we choose not to set any of these values to zero. Therefore, the sparsity of the output result can be different than expected.



Parameters	
$r(T)$	5
$c(T)$	3
q_{r_E}	0.8
q_{r_R}	0.4
q_{c_E}	0.66
q_{c_R}	0.66
t_T	0.4
p_{c_E}	0.5

E

	0	2
0	4	9
0	4	4
0	4	4
2	0	1

R_1

	0	1
0	4	4
1	4	2

T

	0	1	2
0	4	9	9
0	4	4	4
0	4	4	4
1	4	2	
2	0		1

FIGURE 5.9: The DI scenario shown in Figure 5.8 extended with sparsity.

5.7 Row and Column Mappings

We generate row and column mappings for each data scenario we generate. Row mappings are described in indicator matrices **I**, while column mappings are described in mapping matrices **M**. We provide the user with two options: either mappings are written to files in a JSON structure, or they are written directly to SciPy sparse matrices. An example of row and column mappings for the data scenario in Figure 5.9 is shown in Figure 5.10.

$$I_E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad I_R = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad M_E = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad M_R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

FIGURE 5.10: The row and column mappings for the DI scenario shown in Figure 5.9.

5.8 Discussion

It should be noted that there have been previous efforts to create a data generator for the evaluation of data integration scenarios [2], [3]. However, these generators were not suited for our purposes. Specifically, we want to directly be able to control the variables that will be introduced in Chapter **C4**. In addition, other data generators will not be able to generate the indicator and mapping matrices directly. In our implementation of the data generator, we can create the indicator and mapping matrices efficiently, allowing for faster and cheaper experiments.

Contributions discussed in this chapter

– A data generator suitable for generating DI scenarios with target and source redundancy and three different types of joins,
– Automatic generation or row and column mappings corresponding to each DI scenario.

Chapter 6

Cost Estimation

This chapter describes our contribution **C4**, the design and implementation of a cost estimation procedure for Amalur. We estimate in which scenarios we should perform factorized learning and when we should perform learning over materialized data. In this chapter, we aim to answer research question **RQ3**: 'How can we effectively perform cost estimation in Amalur?'. Cost estimation is performed after the user has selected data and an ML model. It takes into account dependent variable y, independent variables X (in the form of a normalized matrix), and ML model m to determine whether a join should be materialized or whether we will perform factorization.

The chapter is structured as follows. First, we describe our hypothesis on the influence of different types of redundancy on the speedups which can be obtained through factorization in Section 6.1. Section 6.2 describes our hypothesis of how sparsity influences speedups. In Section 6.3 we describe the complexity analysis for each of the LA operations and ML models included in Amalur. Lastly, in Section 6.4 we detail the design and implementation of the cost estimation procedure.

6.1 Redundancy

To highlight the effects of source and target redundancy on the speedups achievable through factorized learning, we refer to Figure 6.1. Target redundancy occurs when n rows in table E are related to the same row in a table R_k and $n > 1$. The values in this row in R_k occur n times in T. In Figure 6.1, values in the target table which are redundant according to this definition are marked in blue. We hypothesize that redundancy in the target table is related to more significant speedups achievable through factorization.

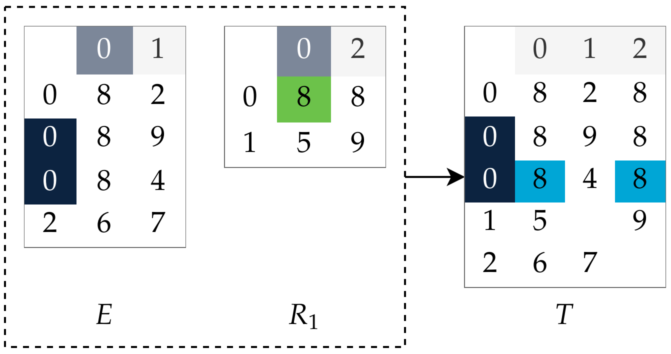


FIGURE 6.1: A data integration scenario illustrating source and target redundancy.

Within the context of factorized learning we introduce a new concept called source redundancy. Source redundancy occurs when one column is present in multiple source tables and there are row matches between these tables. Both source tables then store the same data in this column for the matching rows. In Figure 6.1, values in the source tables which are redundant according to this definition are marked in green. In the target table, this duplication is not present. Our hypothesis states that redundancy in the source tables could be related to slower speedups when applying factorization.

6.2 Sparsity

We inspect what sparsity looks like in Figure 6.2. Here, the sparsity of table E, denoted p_S, equals 6, as there are six cells with zeroes in this table. The sparsities of tables E, R_1, and T are $p_E = 6$, $p_{R_1} = 2$ and $p_T = 11$. The sparsity is made up of two types of sparsity: sparsity resulting from the data and sparsity resulting from the join. Sparsity resulting from the data happens when data explicitly contains zero values. In our implementation, these explicit zeros are not stored in the sparse matrices. Sparsity resulting from the join occurs in the target table. These missing values are also not stored in the sparse materialized table. As missing values and explicit zeroes are not stored, the computations we perform on sparse matrices involve less operations than if we were to perform them on a dense matrix of the same size. Therefore, sparsity should be taken into account when performing cost estimation. The upcoming section details how sparsity is included in the cost estimation procedure.

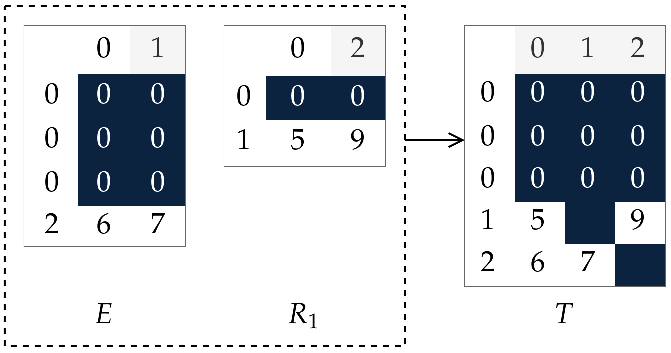


FIGURE 6.2: A DI scenario illustrating sparsity.

6.3 Complexity Analysis

In this section, we describe our complexity analysis for LA operations and for ML models. The complexity of LA operations is defined based on the data characteristics of T, and the complexity of the ML models uses the complexities of the LA operations.

6.3.1 Complexity of linear algebra operations

First, we perform a complexity analysis, characterizing complexity through the number of arithmetic computations for each operation. For the standard case, we consider the number of computations involved in performing an LA operation on the materialized table. For the factorized case, we consider the number of operations involved in performing an LA operation on each source table. Since we are dealing with possibly sparse data, we include this in our complexity analysis. To include sparsity estimations in matrix multiplication, we use adapted computations for matrix multiplication. If we are multiplying two matrices A and B, then the complexity is $O(c_B \cdot nnz(A) + r_A \cdot nnz(B))$, where $nnz(A)$ is the number of nonzero elements in matrix A. We also note the complexity of transposed operations. As described in Section 4.4, $T^T X$ is actually defined as $(X^T T)^T$ and XT^T is actually defined as $(TX^T)^T$. Here, we ignore the complexity added by the transposes, as these are not executed on normalized matrices. The resulting complexity analysis is shown in Table 6.1.

Operation	**Standard**	**Factorized**
$T \oslash x$ $^T T \oslash x$ $f(T)$ $f(T^T)$ rowSums(T) rowSums(T^T) colSums(T) colSums(T^T)	$nnz(T)$	$\sum_{k=1}^{K} nnz(S_k)$
TX	$c_X \cdot nnz(T) + r_T \cdot nnz(X)$	$\sum_{k=1}^{K} c_X \cdot nnz(S_k) + r_{S_k} \cdot nnz(X)$
$T^T X$	$c_T \cdot nnz(X) + c_X \cdot nnz(T)$	$\sum_{k=1}^{K} c_{S_k} \cdot nnz(X) + c_X \cdot nnz(S_k)$
XT	$c_T \cdot nnz(X) + r_X \cdot nnz(T)$	$\sum_{k=1}^{K} c_{S_k} \cdot nnz(X) + r_X \cdot nnz(S_k)$
XT^T	$r_X + nnz(T) + r_T \cdot nnz(X)$	$\sum_{k=1}^{K} r_X + nnz(S_k) + r_{S_k} \cdot nnz(X)$

TABLE 6.1: Arithmetic computations of the standard and factorized Linear Algebra operations.

6.3.2 Computing the sparsity of T

Some elements in the complexity analysis can be inferred directly from the source tables, while others require estimation. Elements which can be directly inferred include r_X, c_X, r_{S_k}, c_{S_k} and $nnz(S_k)$, as these are stored in SciPy objects. The elements r_T and c_T cannot be inferred from the source tables directly, but can be inferred from the indicator matrices and mapping matrices, respectively. The element that requires the most complicated computations is $nnz(T)$, which depends upon the source tables' sparsities, the indicator matrix and the mapping matrix. In order to compute $nnz(T)$, we compute T using the materialization computation defined in section 4.4.6. Then, we compute the sparsity directly from the result of this computation.

6.3.3 Complexity of Machine Learning models

Using the complexity definitions for LA operations, we estimate the complexity of ML models. Here, we only consider the operations in ML model algorithms that involve our rewrites.

Linear and logistic regression We define the complexity of linear and logistic regression on data T below. The shape of the weights vector w is $c_T \times 1$ and we assume it is dense, so $nnz(w) = r_w \times c_w$. Matrix X denotes an intermediate result in the linear regression algorithm, of which the size is $r_T \times 1$. We assume this intermediate result is also dense, so $nnz(X) = r_X \times c_X$. First, we define the complexity in the standard case.

$$O_{\text{standard}}(T) = \underbrace{c_w \cdot nnz(T) + r_T \cdot nnz(w)}_{Tw} + \underbrace{c_T \cdot nnz(X) + nnz(T)}_{T^T X}$$

Next, we define the complexity of the factorized case.

$$O_{\text{factorized}}(T) = \underbrace{\sum_{k=1}^{K} c_w \cdot nnz(S_k) + r_{S_k} \cdot nnz(w)}_{Tw} + \underbrace{\sum_{k=1}^{K} c_{S_k} \cdot nnz(X) + nnz(S_k)}_{T^T X}$$

Gaussian NMF We define the complexity of Gaussian NMF on data T below. From the parameter r supplied to Gaussian NMF we can infer the shapes of matrices W and H. The shape of matrix W is $r_T \times r$, the shape of matrix H is $c_T \times r$. We assume both matrices are dense, so $nnz(W) = r_W \times c_W$ and $nnz(H) = r_H \times c_H$. The complexity in the standard case is shown below.

$$O_{\text{standard}}(T) = \underbrace{c_T \cdot nnz(W) + c_W \cdot nnz(T)}_{W^T T} + \underbrace{r_H \cdot nnz(T) + r_T \cdot nnz(H)}_{TH^T}$$

The complexity in the factorized case is shown below.

$$O_{\text{factorized}}(T) = \underbrace{\sum_{k=1}^{K} c_{S_k} \cdot nnz(W) + c_W \cdot nnz(S_k)}_{W^T T} + \underbrace{\sum_{k=1}^{K} r_H \cdot nnz(S_k) + r_{S_k} \cdot nnz(H)}_{TH^T}$$

K-Means We define the complexity of K-Means on data T below. From the parameter k supplied to K-Means we can infer the shapes of C and A. The shape of C is $c_T \times k$, the shape of A is $r_T \times k$. We assume both C and A are dense, so $nnz(C) = r_C \times c_C$ and $nnz(A) = r_A \times c_A$. The complexity in the standard case is shown below.

$$\begin{aligned} O_{\text{standard}}(T) = \underbrace{2nnz(T)}_{\text{rowSums}(T^2)} &+ \underbrace{nnz(T) + c_C \cdot nnz(T) + r_T \cdot nnz(C)}_{(2\times T)C} \\ &+ \underbrace{c_T \cdot nnz(A) + c_A \cdot nnz(T)}_{T^T A} \end{aligned}$$

The complexity in the factorized case is shown below.

$$O_{\text{factorized}}(T) = \underbrace{2\sum_{k=1}^{K} nnz(S_k)}_{\text{rowSums}(T^2)} + \underbrace{\sum_{k=1}^{K} nnz(S_k) + c_C \cdot nnz(S_k) + r_{S_k} \cdot nnz(C)}_{(2\times T)C}$$
$$+ \underbrace{\sum_{k=1}^{K} c_{S_k} \cdot nnz(A) + c_A \cdot nnz(S_k)}_{T^T A}$$

6.3.4 Complexity ratio

To compute whether the complexity of the standard approach is larger than the complexity of the factorized approach, we calculate the complexity ratio. The complexity ratio is defined as the standard complexity divided by the factorized complexity. In theory, a complexity ratio above one indicates that a speedup should occur. However, we expect that due to overhead this will not (always) be true. In Section 6.5 we show for which complexity ratios we can expect speedups.

6.4 Design

The design of our cost estimation approach is shown in Figure 6.3. We make the following assumptions in the design of the cost estimator. First, we assume we have been provided the source tables, indicator matrix and mapping matrix. We argue that we can make this assumption as our data is obtained from a system for data integration [8]. In addition, we assume we have been provided with a user-selected ML model, including model hyperparameters.

Figure 6.3 shows the cost estimation procedure from top to bottom. The input to the cost estimation procedure are the ML model and normalized matrix, components which are also shown in the general architecture of Amalur in Figure 4.1. To perform cost estimation for any data integration scenario, we use data characteristics inferred from the normalized matrix. These characteristics are the dimensions and sparsities of the source tables and the dimensions and sparsity of the materialized table. In addition, we consider the ML model and model hyperparameters selected by the user. From the data characteristics and hyperparameters we calculate the complexity of ML models, which are sums of LA operation complexities as described in Section 6.3.3. From these sums, we calculate the complexity ratio. If the complexity ratio is higher than threshold τ, we estimate that factorization should be applied. The determination of τ is explained in the next section.

6.5 Calibrating the Decision Threshold

To predict whether speedups will occur, we have to set the decision threshold τ such that it distinguishes cases where we do not expect speedups through factorization from cases where we do. Initially, we wanted to base our prediction solely on the complexity ratio. However, this led to falsely predicted speedups for small datasets. Therefore, we also consider the size of the dataset by incorporating the size of the materialized table T ($r_T \times c_T$) in our cost estimation. To determine the decision boundary we generate synthetic datasets using our data generator. The details of these synthetic datasets are described in Section 7.2.2. As we create datasets

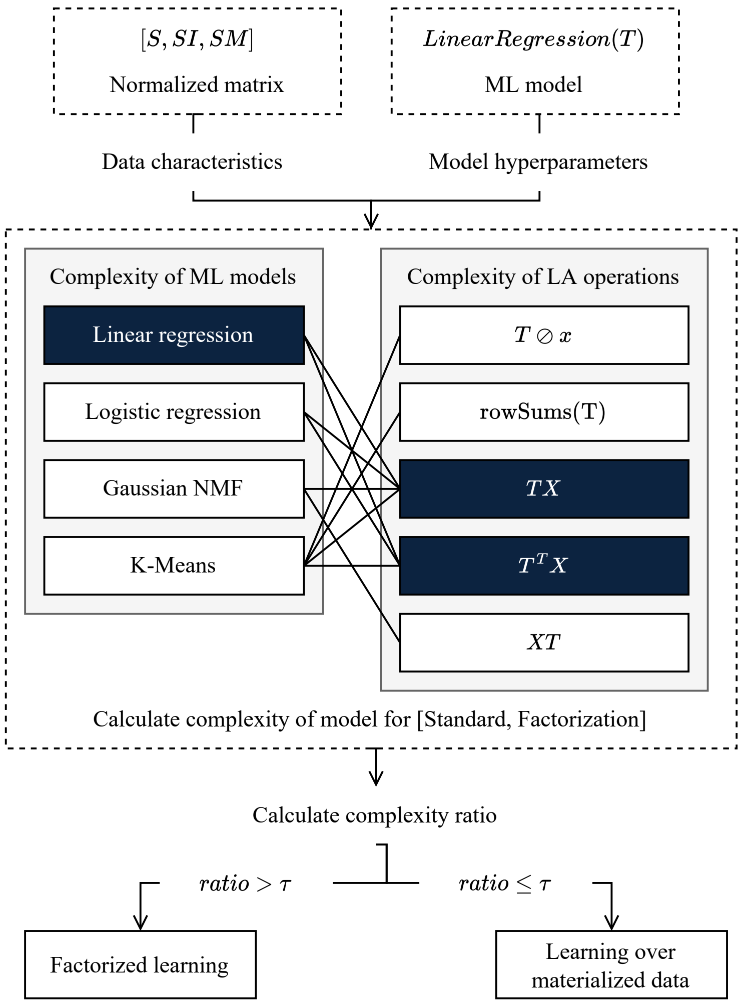


FIGURE 6.3: The design of the cost estimator for Amalur.

with varying complexity ratios for scalar operations, there are some combinations of complexity ratio and dataset size missing from the experiments. Then, we compute the lowest achieved speedup for each combination of the complexity ratio and the size of T. The results are visualized in Figure 6.4 for LA operations.

In Figure 6.5, we see the result of the same experiments for ML models. We use this figure to determine our decision boundary, as we want to perform accurate cost estimation on ML models. We want to make decisions conservatively, preferring false negatives over false positives. This is why inspecting the lowest achieved speedup is a better choice than inspecting the mean or maximum speedups. Theoretically, the plots should be the same for all ML models. The plots are very similar, but not the same. Overall, we see that speedups for data sizes below 200K are less than one or relatively small. For data sizes above 200K, we notice that significant speedups can be achieved for complexity ratios higher than 1.5. Therefore, we set our decision threshold τ to 1.5, meaning that a complexity smaller than or equal to 1.5 indicates that we should not apply factorization, while a complexity ratio higher than 1.5 indicates we should factorize. For datasets smaller than 200K entries, we should not apply factorization.

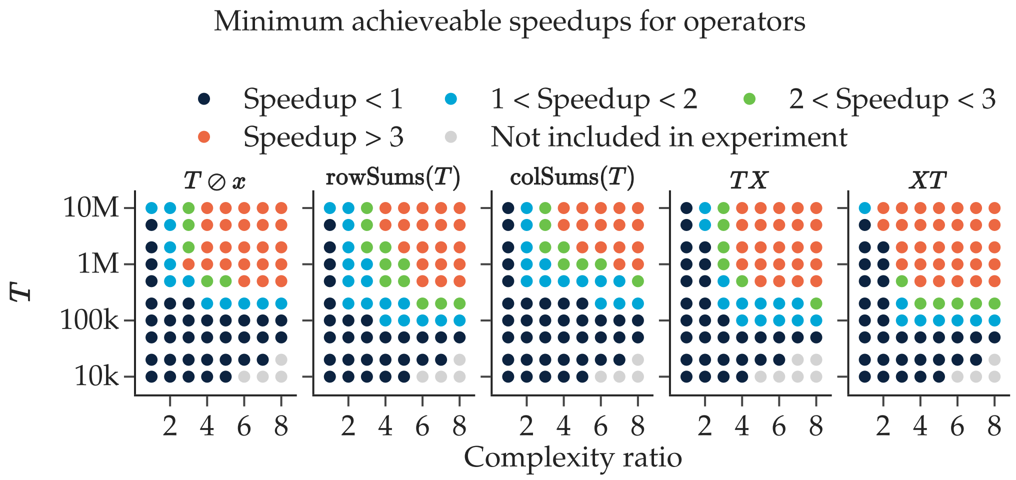


FIGURE 6.4: Minimum achieved speedups for LA operations relative to complexity ratio and the size of T.

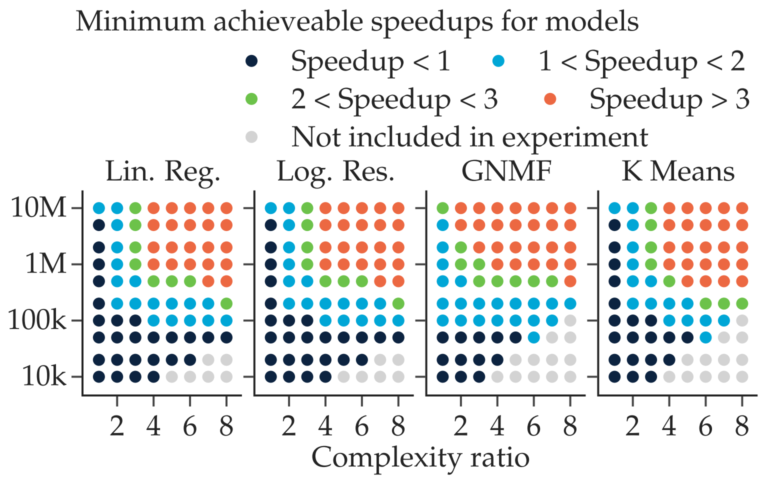


FIGURE 6.5: Minimum achieved speedups for ML models relative to complexity ratio and the size of T.

6.6 Discussion

In this chapter, we detailed our cost estimation procedure. This cost estimation procedure is a step forward from previous efforts in two ways. The first advantage of our cost estimation procedure is that it can handle both row overlap and column overlap and is applicable to any join or any sequence of of joins. The cost estimation also works for unions. The second advantage is that our cost estimation procedure is designed in such a way that it estimates the cost of individual operations and is tailored to each ML model.

Contributions discussed in this chapter

– The identification of source and target redundancy,

– A flexible cost estimation procedure suitable for any DI scenario covered by Amalur and tailored to individual ML models, which can easily be extended to include other ML models.

Chapter 7

Evaluation

In this chapter, we detail our evaluation. In Section 7.1 we describe the real-life datasets we will use in our experiments. In Section 7.2 we describe the setup of our experiments. This includes editing our real datasets, generating synthetic data, and software and hardware specifications. Then, we evaluate the factorized learning we implemented in Amalur in Section 7.3. In Section 7.4 we evaluate our main contribution, our cost estimation procedure.

7.1 Datasets

dataset	r_T	c_T	$nnz(T)$	n_S	$nnz(S)$	r_S	c_S
book*	253120	81663	2024960	2	83628	27876	28022
					249860	49972	53641
expedia	942142	52282	28263069	3	5652852	942142	27
					107451	11939	12013
					555315	37021	40242
flight	66548	13669	1385834	4	55301	66548	20
					3240	540	718
					22169	3167	6464
					22190	3170	6467
lastfm*	343747	55252	4468711	2	39992	4999	5019
					250000	50000	50233
movie*	1000209	13348	27005643	2	30200	6040	9509
					81532	3706	3839
walmart	421570	2441	5901781	3	421570	421570	1
					23400	2340	2387
					135	45	53
yelp*	215879	55606	8635160	2	380655	11535	11706
					307111	43873	43900

TABLE 7.1: Characteristics of the Hamlet [26] datasets. Datasets for which the entity table is not included in the normalized matrix are indicated using $*$.

To evaluate Amalur for PK-FK joins, we use datasets developed for Project Hamlet[1] [26], [35]. These are seven real-world datasets used in experiments for state-of-the-art systems [5], [26], [31], [35]. The datasets consist of two or more tables and are connected in a star schema. For some datasets, the entity table has no columns except for foreign keys. Since the keys are dropped from the table in the normalized

[1]Project Hamlet datasets: `https://adalabucsd.github.io/hamlet.html`

matrix, these entity tables have no columns and are therefore not included in the table characteristics. We indicate these tables using an asterisk ($*$). The characteristics of these datasets are described in Table 7.1.

The datasets were originally collected from different sources. The datasets Walmart, Expedia and Yelp were found in Kaggle[2], datasets MovieLens1M and BookCrossing were obtained from GroupLens[3], the dataset Flights was found on openflights[4], the LastFM dataset was created from two sources, the lastfm API[5] and the lastfm website[6]. Categorical variables were converted into one-hot encodings. All of these datasets are structured according to a star schema. We describe the schemas of each of these datasets, which were previously described by [26], in Appendix A.

7.2 Experiment Setup

We run our experiments with 20 AMD EPYC 7H12 64-Core Processor CPUs, 75 GB RAM and a 4TB disk. The system uses Ubuntu 20.04.4 LTS as the OS. Each experiment is run on one CPU. Our experiments have been written in Python 3.10.4. We use SciPy version 1.8.0 and NumPy version 1.22.4. We set the number of iterations of ML models to 20.

7.2.1 Real datasets

The Hamlet datasets were originally only used to evaluate inner joins. To be able to evaluate meaningful left outer and full outer joins, we remove rows from selected tables in each dataset. For an inner join, we remove data from one of the attribute tables. For the Expedia dataset we remove data from table S_3 *Searches*. For the Flights dataset we remove data from table S_4 *destAirports*. For the datasets Movies, LastFM, Yelp and Books we remove data from the table *Users* in each dataset. For an outer join, we remove data from the same tables as the inner join, and additionally, we remove data from the S_1 of each dataset, which, in each dataset, is the entity table.

7.2.2 Synthetic datasets

In order to generate synthetic datasets, we use our data generator. We generate one table E and one table R, determined by $n_R = 1$. We generate datasets where the number of rows in target table T, $r_T \in [10, 20, 50, \ldots, 10000, 20000, 50000]$, the number of columns $c_T = [100, 1000]$, the number of columns in E as a percentage of the number of columns in T, $q_{c_E} = 0.1$ and the sparsity of E, $p_E = 0.1$. In addition, we vary the number of columns in R as a percentage of the number of columns in T, q_{c_R}, from $1 - q_{c_E}$ for no column overlap to $1 - q_{c_E} + 0.1$ for column overlap. We generate three types of joins: the inner join, left join and full outer join. For a left outer join, parameter $t_R = 0.9$. For a full outer join, in addition to varying parameter t_R, we fix the number of rows in E as a a percentage of the number of rows in T, $q_{r_E} = 0.9$. We then generate datasets for complexity ratios in the range $[1, 2, ..., 8]$ and calculate the remaining parameters for the data generator.

[2]Kaggle: `www.kaggle.com`
[3]GroupLens: `www.grouplens.org`
[4]openflights: `www.openflights.org`
[5]lastfm API data: `mtg.upf.edu/node/1671`
[6]lastfm: `www.last.fm`

7.3 Factorization in Amalur

First, we evaluate the performance of our implementation of Amalur. To paint a complete picture, we evaluate both individual LA operations as well as ML models on both synthetic data and real datasets.

7.3.1 Linear algebra operations

We evaluate each LA operation in the factorized case against the non-factorized case. The non-factorized case uses basic NumPy operations on the materialized data. To evaluate multiplication operations, we need to create a matrix X. For LMM, we create a dense matrix of size $c_T \times 100$. For RMM, we create a dense matrix of size $100 \times r_T$.

Real datasets

In Figure 7.1 we see the operation time of different LA operations for each Hamlet dataset. Each type of join has a different number of rows in a selected set of source tables as described in section 7.2.1. In general, we see that left matrix multiplication (TX) and right matrix multiplication (XT) have significantly longer runtimes than scalar operations ($T \oslash x$), column summation (colSums(T)) and row summation (rowSums(T)). The operation time for right matrix multiplication is slightly lower on average than the operation time of left matrix multiplication. This makes sense, as T has more rows than columns, so X is larger in XT than in TX.

The figure shows that **speedups occur for most datasets and operations**. Some datasets show high speedups, while others show relatively lower speedups. The datasets with the smallest speedups are the books dataset, the flight dataset, and the walmart dataset. The dataset with the highest speedups is the yelp dataset. In order to investigate the speedups on transposed data, we inspect Figure B.1 in Appendix B. This figure shows the operation times of each operation on T^T. The results are very similar to those shown in Figure 7.1, indicating that our implementation of the transpose is effective.

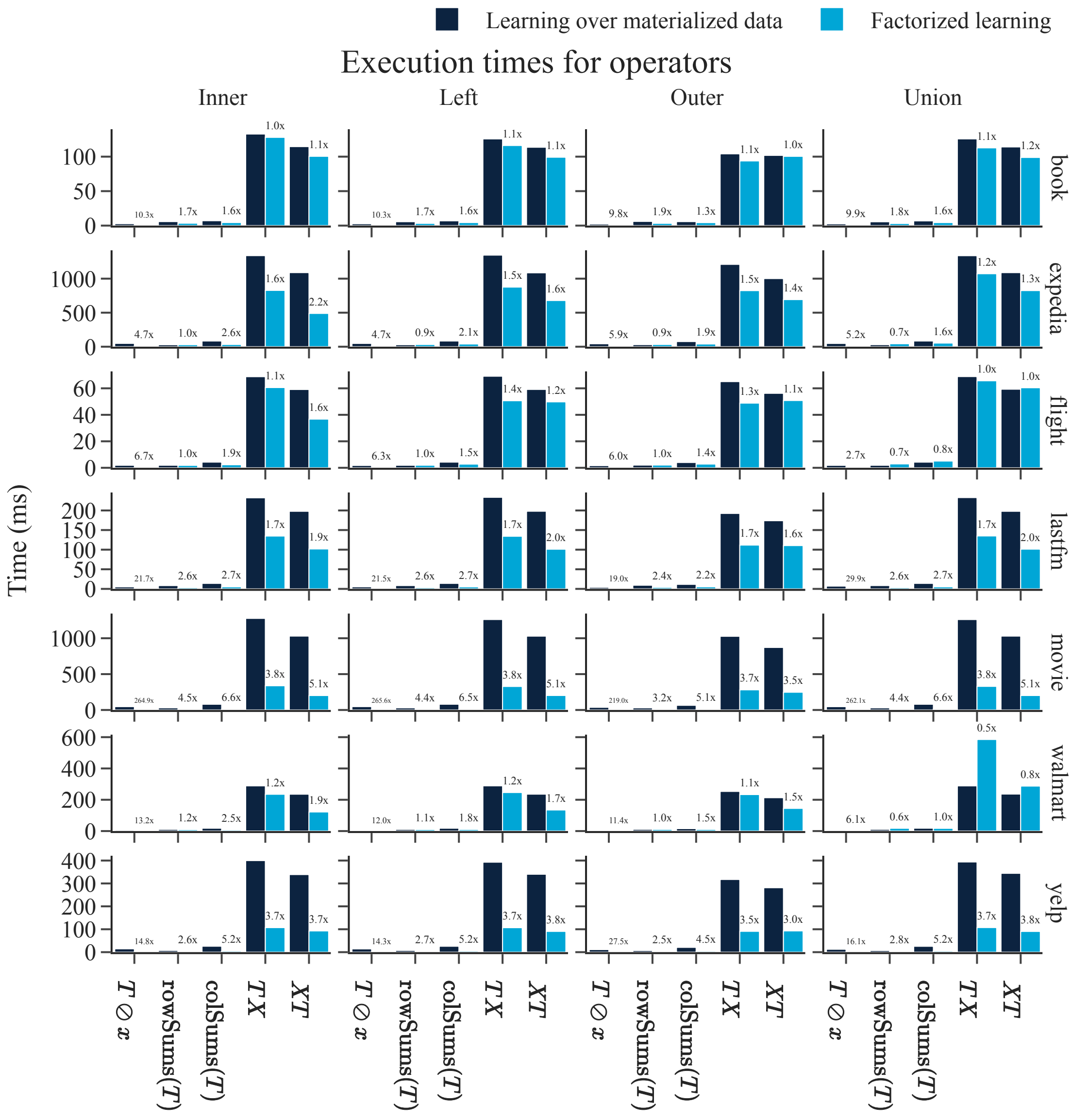


FIGURE 7.1: Evaluation of Amalur LA operations on Hamlet dataset.

Synthetic datasets

Since we hypothesize that the complexity ratio can be used to explain speedups, we generate datasets with varying complexity ratios. Our generation method is explained in Section 7.2.2. We inspect the influence of the complexity on the achieved speedup in Figure 7.2. Figure 7.2 shows each linear algebra operation's mean speedup and the standard deviation around this mean. The dotted line indicates the point at which the speedup equals one. In any case where the speedup is smaller than one, materialization should be performed. If the speedup is greater than one, factorization should be performed. We can achieve speedups on all linear algebra operations. For all operations we observe a **positive correlation between the complexity ratio and the mean speedup**.

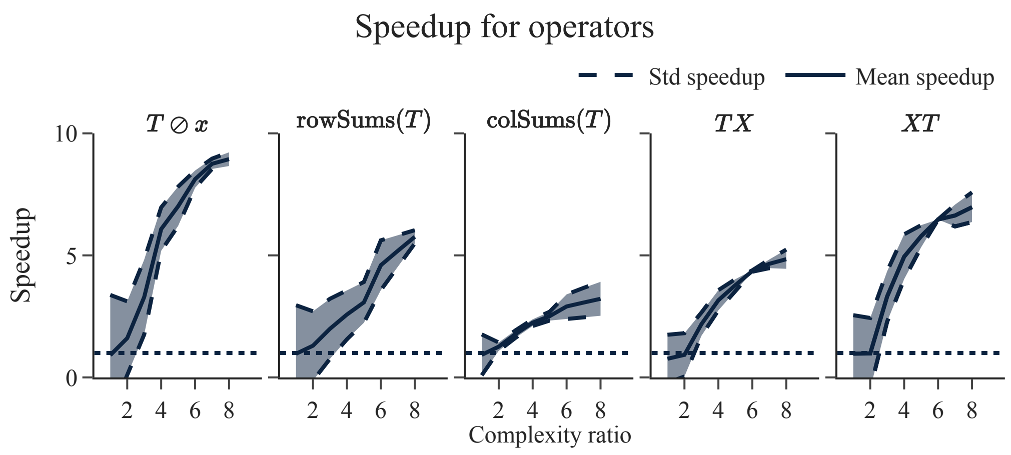


FIGURE 7.2: Evaluation of Amalur LA operations on synthetic datasets.

7.3.2 Machine Learning Models

We evaluate each ML model in the factorized case to compare it to its non-factorized alternative. Again, we record the time needed for materialization in the baseline case and the time needed for the construction of the data representation in the factorized case.

Real datasets

The results are shown in Figure 7.3. Each bar shows the time it takes to perform twenty iterations of a model for a dataset. The dataset books has only slowdowns for factorized ML models. The datasets expedia, flights, lastfm and walmart have some speedups and some slowdowns. The datasets movie and yelp have only speedups. Overall, the performance of factorization is lower for the union scenarios. The performance of the baseline remains roughly the same across different join conditions.

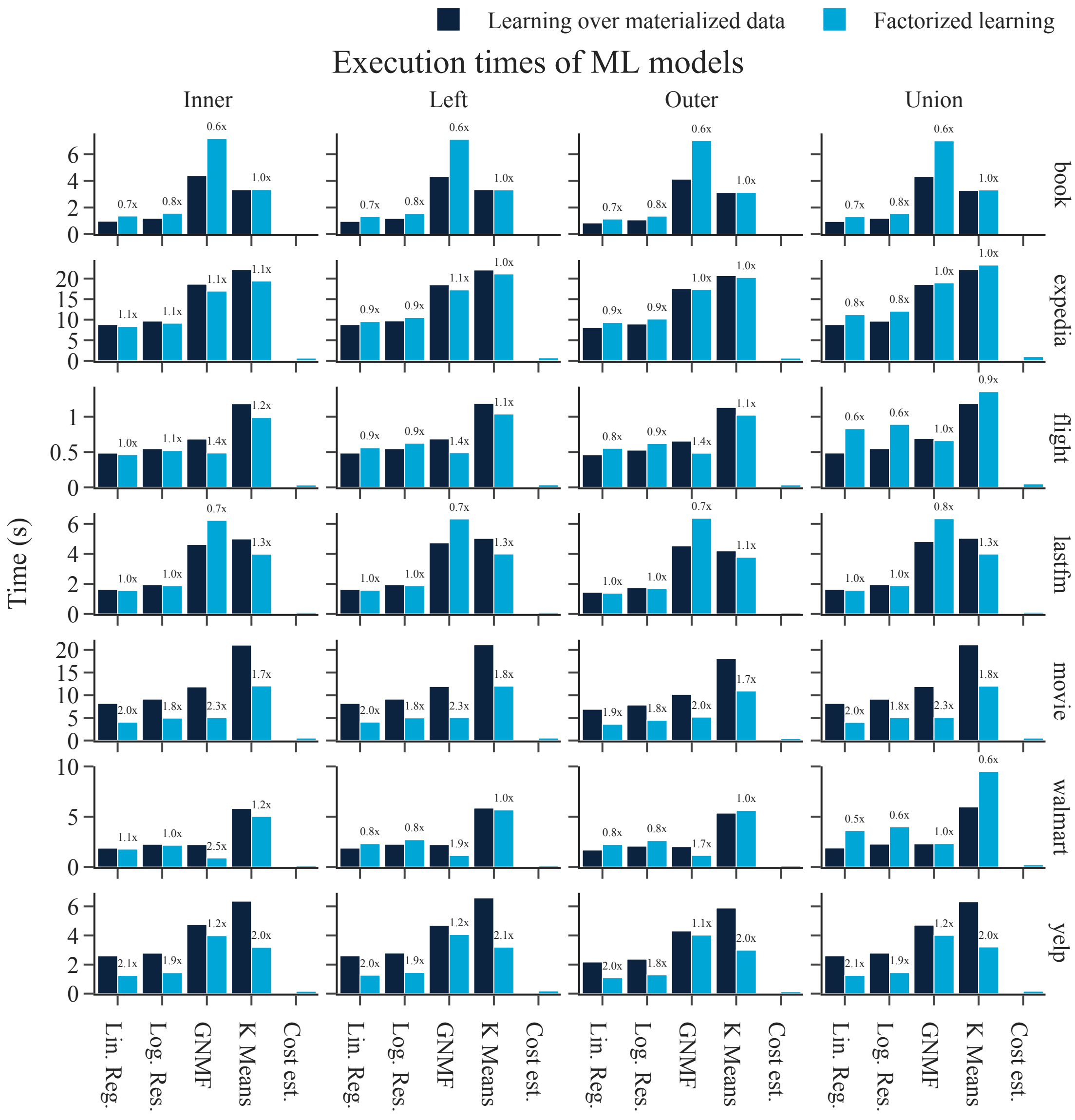


FIGURE 7.3: Evaluation of Amalur ML models with the number of iterations = 20 on Hamlet dataset. Annotations show the speedup of factorized learning compared to learning over materialized data.

Synthetic datasets

We inspect the influence of the complexity on the achieved speedup for different ML models in Figure 7.4. The pattern shown in Figure 7.2 holds in this figure. We see **larger speedups for larger complexity ratios**. In addition, the figure suggests that the standard deviation of the speedup is larger for lower complexity ratios.

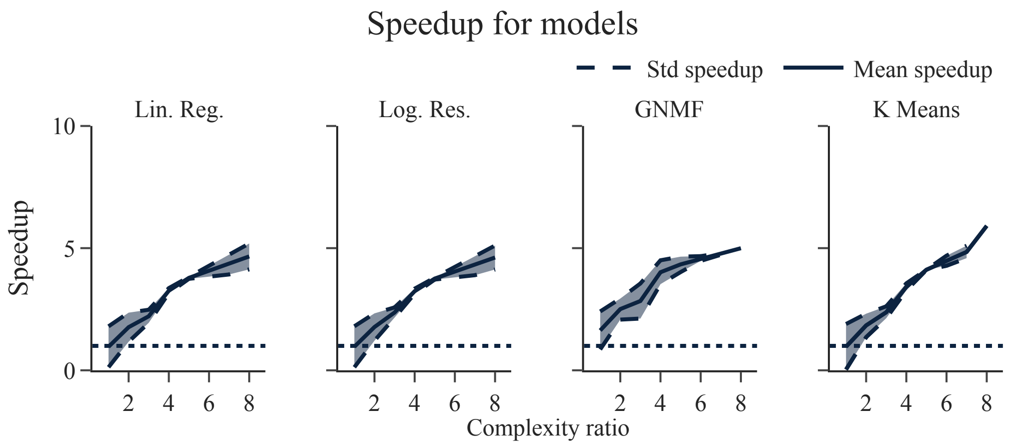


FIGURE 7.4: Evaluation of Amalur ML models on synthetic datasets.

7.4 Cost Estimation

This section describes the evaluation of our cost estimation model. We expect that Morpheus' cost estimation works well for scenarios representing inner joins with target redundancy. We also expect, however, that our cost estimation will create better predictions than Morpheus', as it takes into account sparsity and is tailored to individual LA operations and ML models.

7.4.1 Methodology

As our baseline, we choose the cost estimation procedure defined by Morpheus [5], which considers the tuple ratio ($TR = \frac{r(S)}{r(R)}$) and feature ratio ($FR = \frac{c(S)}{c(R)}$). We evaluate our cost estimation and the cost estimation used by Morpheus on the real datasets and synthetic datasets. These datasets are described in sections 7.2.1 and 7.2.2, respectively.

Calibrating the baseline

To create a fair comparison between our cost estimation and Morpheus' cost estimation, we define thresholds for Morpheus in the same way we defined thresholds for our method in Section 6.5. We use our data generator to create datasets with $r_T = r_S = 10000$, $c_S = 20$, $n_R = 1$, and $p_S = 0$, where there is no column overlap between tables and the target table is the result of an inner join between S and R_1. Then, we vary $TR \in [0, 1, ..., 20]$ and $FR \in [1, 1.2, 1.4, ..., 4]$. We record the speedups of LA operations and ML models and calculate the lowest achieved speedup for each combination of FR and TR. The results for the LA operations are shown in Figure 7.5.

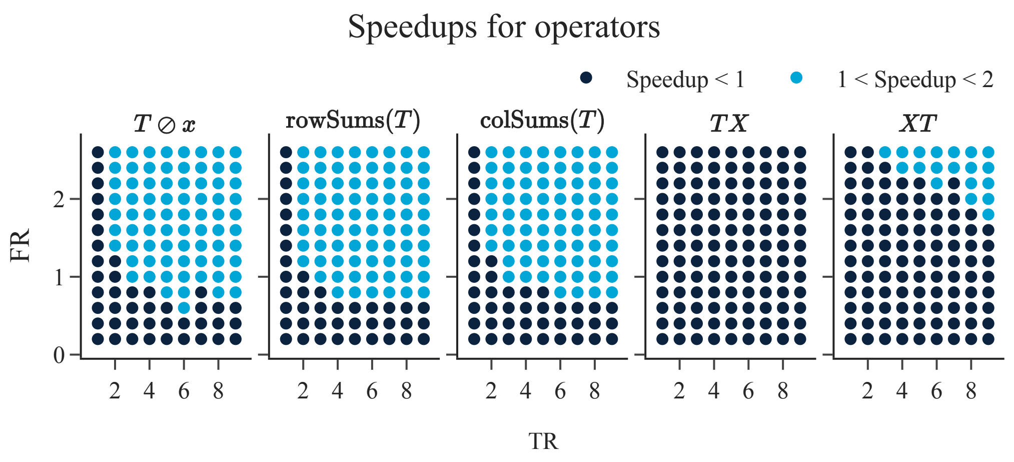


FIGURE 7.5: Minimum achieved speedup for LA operations relative to the tuple ratio (TR) and feature ratio (FR).

The results for the ML models are shown in Figure 7.6. Keeping in line with the calibration of our cost estimation procedure, we use the analysis on ML models to determine our decision boundary, as we want to perform accurate cost estimation on ML models. Overall, we determine that **speedups are more likely to occur for larger tuple ratios and larger feature ratios**. We determine that when $TR < 4$ or $FR < 0.6$, we apply learning over materialized data. In any other case, we apply factorized learning.

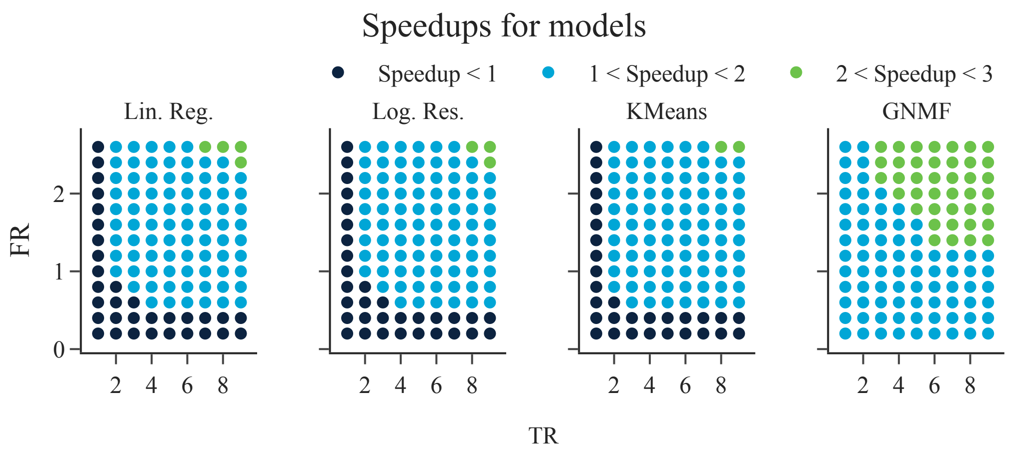


FIGURE 7.6: Minimum achieved speedup for ML models relative to the tuple ratio (TR) and feature ratio (FR).

In addition to determining thresholds using TR and FR, we also define that any datasets where $r_T \times c_T < 200K$ are predicted to have no speedups by Morpheus. In this way, we can fairly compare our approach and Morpheus' approach despite performing an evaluation on relatively small datasets.

7.4.2 Linear Algebra operations

First, we inspect the performance of the two cost estimation procedures on LA operations. We inspect the performance of cost estimation on both our real and synthetically generated datasets.

Real datasets

Figure 7.7 shows confusion matrices for Morpheus' cost estimation on the Hamlet datasets. These confusion matrices are separated into four groups: one group contains all cases where speedups occurred that were correctly predicted (true positives, TP), one group contains all cases where speedups did occur which were incorrectly predicted (false negatives, FN), one group contains all cases where speedups did not occur which were correctly predicted (true negatives, TN), and the last group contains all cases where speedups did not occur which were incorrectly predicted (false positives, FP).

We include only the scenarios for which Morpheus' cost estimation approach was designed. In this case, this means we only consider inner joins. As shown in Figure 7.7, for LA operations on our real datasets, there are almost no scenarios where factorization leads to slowdowns. Morpheus incorrectly predicts the slowdown that does occur. The **majority of the scenarios where factorization leads to speedups are incorrectly predicted by Morpheus**.

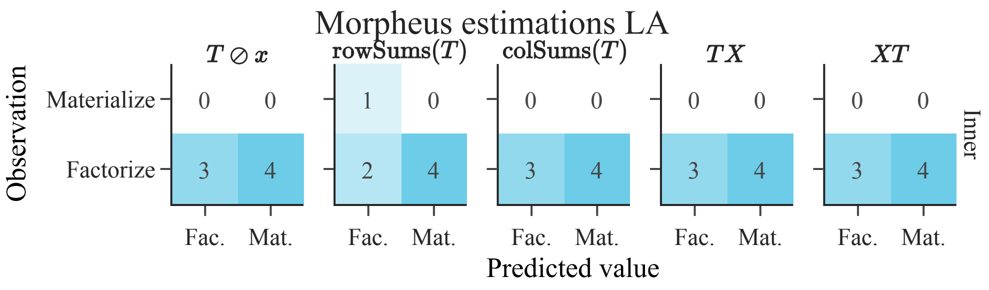


FIGURE 7.7: Confusion matrices for Morpheus' cost estimation of LA operations on the Hamlet datasets.

We can inspect the same results for transposed data. These results are shown in Figure B.2 in Appendix B. Compared to the non-transposed case, the predictions for column summation now include one false positive, while the predictions for row summation no longer include a false positive.

Next, we inspect the performance of our cost estimation procedure. Figure 7.8 shows confusion matrices for our cost estimation on the Hamlet datasets. As we are able to support multiple types of joins and the union, we include the left outer join, full outer join and union cases as well as the inner join cases. In the scenarios where factorized learning results in speedups, **our estimation correctly predicts almost all cases**, except for left matrix multiplication. In the scenarios where factorized learning does not result in speedups, our estimation makes the same mistake as Morpheus' estimation. The performance of our cost estimation is approximately equal across different types of joins except for the union, for which our performance decreases. The results on transposed data are shown in Figure B.3 in Appendix B. Compared to Figure 7.8, we have two more false positives for column summation for each join type, but fewer false positives for row summation. For left matrix multiplication we have more false negatives.

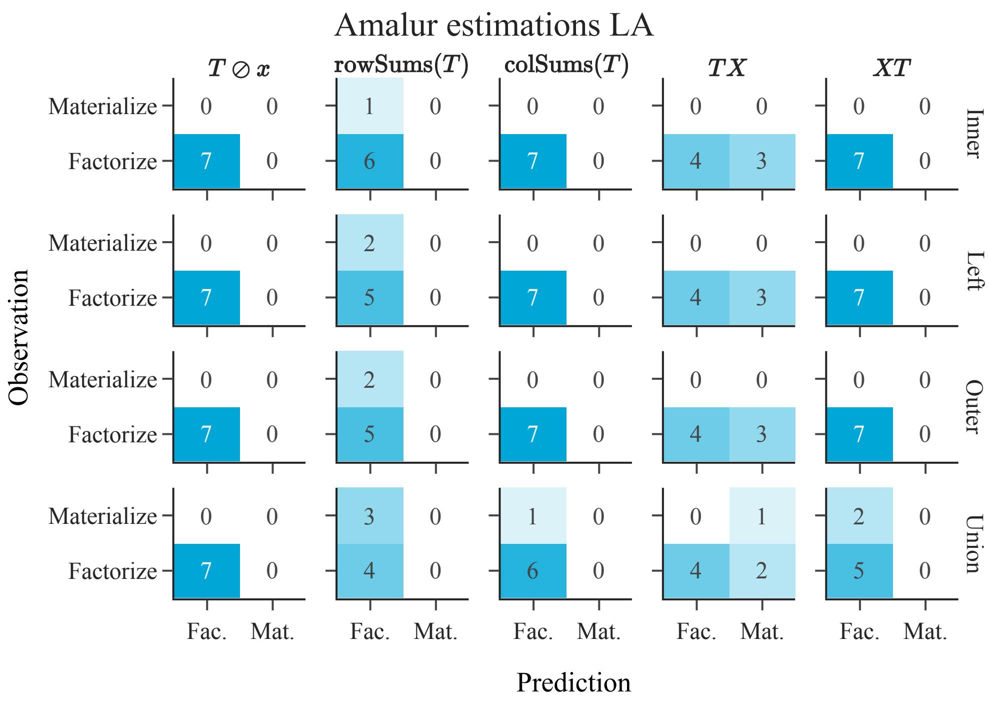


FIGURE 7.8: Confusion matrices for Amalur cost estimation of LA operations on the Hamlet datasets.

Synthetic datasets

To evaluate cost estimation methods on synthetic datasets, we use the datasets we generated according to the description in Section 7.2.2. We categorize these datasets according to the type of join and the types of redundancy present in the dataset. In addition to the number of false and correct predictions investigated in the previous section, we wish to investigate the speedups and slowdowns for falsely predicted and correctly predicted cases. Therefore, we separate our results into four groups: one contains all true positives TP, one part contains all false negatives, FN, one part contains all true negatives, TN, and one part contains all false positives, FP.

The result is shown in Figure 7.9. We consider only scenarios for which Morpheus' cost estimation was designed. This means we consider scenarios that have only target redundancy and which also represent inner joins. Firstly, we can see the largest part of Morpheus' predictions resulted in false negatives ($n = 395$), followed by false negatives ($n = 385$), true positives ($n = 221$) and then false positives ($n = 19$). We calculate the mean speedup of the individual groups for each LA operation, assuming that the speedups follow an normal distribution. We see that the mean speedup for true positives is higher than for false negatives. This suggests that **Morpheus' estimations are more likely to mispredict small speedups than large speedups**. In addition, the mean speedup for false positives is slightly smaller for true negatives. Therefore, it might be the case that **small slowdowns are more likely to be mispredicted by Morpheus' cost estimation than large slowdowns**.

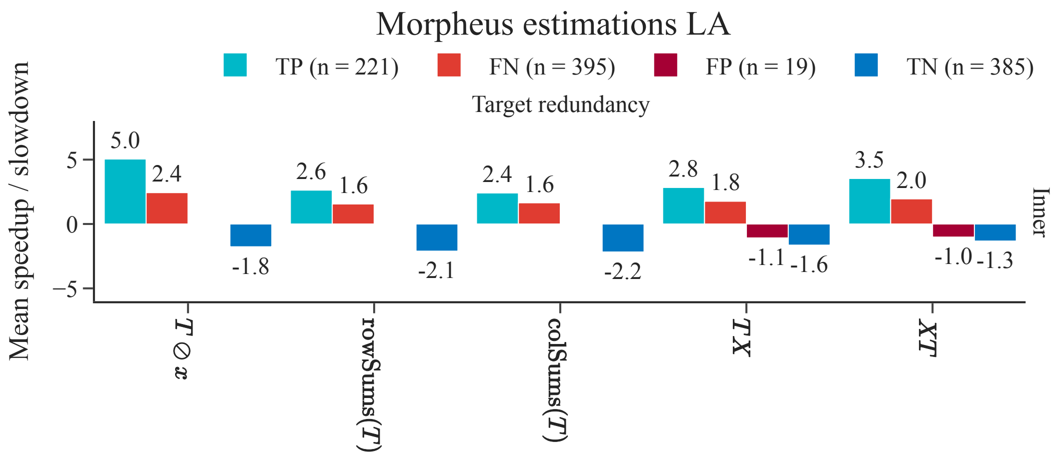


FIGURE 7.9: Mean speedups and slowdowns on LA operations for correctly and incorrectly predicted cases by Morpheus' cost estimation on synthetic datasets.

Now we inspect the performance of our cost estimation procedure. Figure 7.10 shows the relation between the mean speedup or slowdown and the correctness of our cost estimation approach for different types of redundancy and different join conditions. As we include multiple types of redundancy and join conditions, the number of true positives, false negatives, false positives, and true negatives is also larger. The ratios between these measures, however, are approximately the same as in Figure 7.9. The largest parts of our predictions resulted in true positives ($n = 1406$), followed by true negatives ($n = 1207$), false negatives ($n = 921$), and lastly false positives ($n = 46$). For the inner and left join, we see that the mean speedup for true positives is much higher than the mean speedup for false negatives. Therefore, **our cost estimator is more likely to mispredict small speedups than large speedups**. For the outer join we see the opposite effect. for all join conditions we see that the mean slowdown for false positives is occasionally much smaller than that for true negatives. Therefore, **small slowdowns are more likely to be mispredicted by our cost estimation than large slowdowns**.

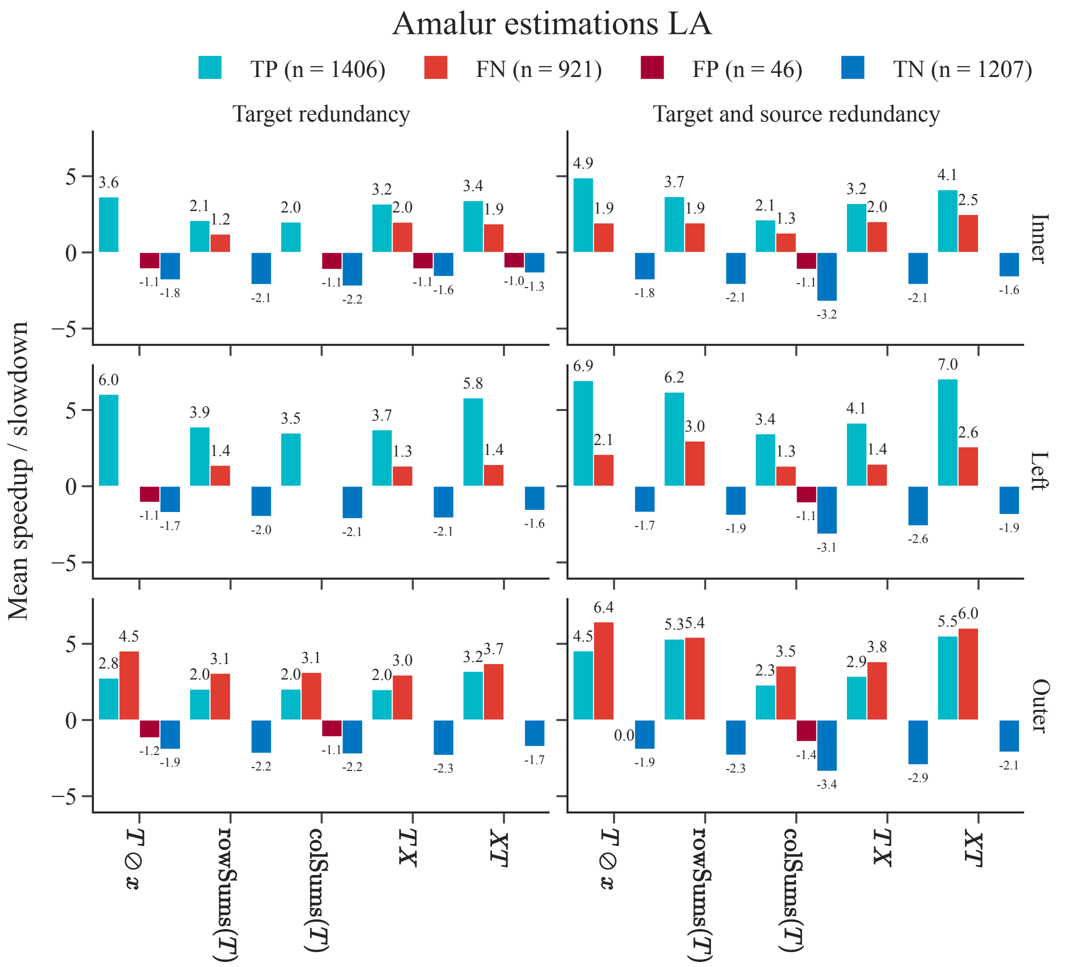


FIGURE 7.10: Mean speedups and slowdowns on LA operations for correctly and incorrectly predicted cases by our cost estimation on synthetic datasets.

7.4.3 Machine Learning models

Now, we inspect the performance of the two cost estimation procedures on ML models.

Real datasets

Figure 7.11 shows confusion matrices for Morpheus' cost estimation on the Hamlet datasets. With no false positives, **Morpheus' cost estimation performs well for the real datasets**. The predictions for GNMF are the most accurate out of the ML models, the predictions for K-Means are the least accurate.

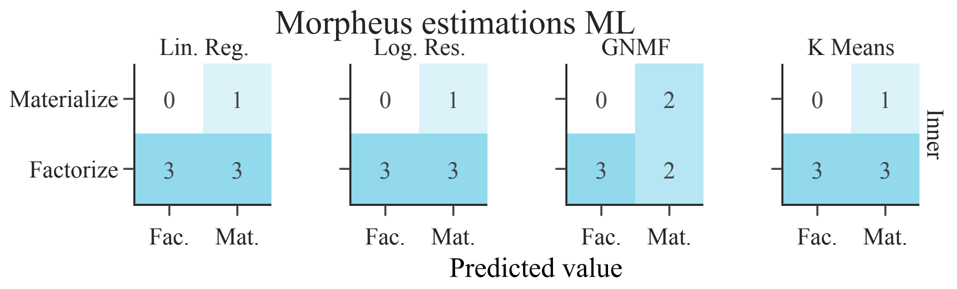


FIGURE 7.11: Confusion matrix for Morpheus cost estimation of ML models on Hamlet datasets.

Figure 7.12 shows confusion matrices for Amalur. The performance of Amalur differs primarily between ML models. Our approach creates **accurate predictions with some false positives for linear and logistic regression**, but shows **less accurate predictions for GNMF**. The predictions on K-Means are reasonably accurate with no false positives.

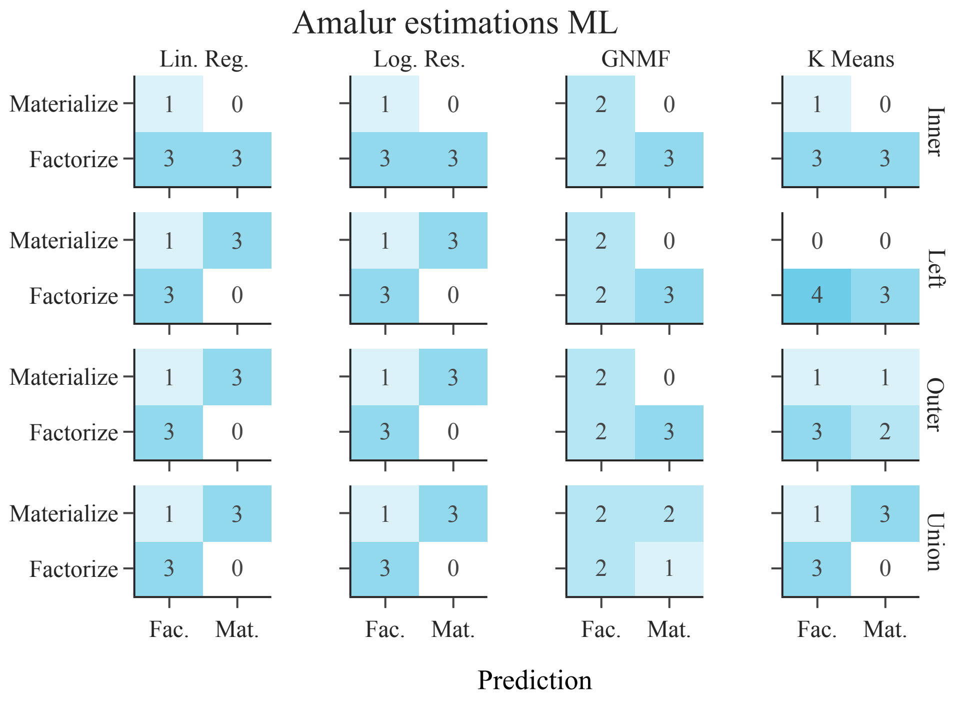


FIGURE 7.12: Confusion matrix for Amalur cost estimation of ML models on Hamlet datasets.

Synthetic datasets

We also inspect the performance of cost estimation methods for ML models on synthetic datasets. The results for Morpheus' cost estimation are shown in Figure 7.13. Again, we see that **the mean speedup for true positives is higher than the mean speedup for false negatives**. We do not see any false positives ($n = 0$), although we see more false negatives ($n = 343$) than true negatives ($n = 281$). The number of true positives ($n = 192$) is equal to the number of cases where speedups occur.

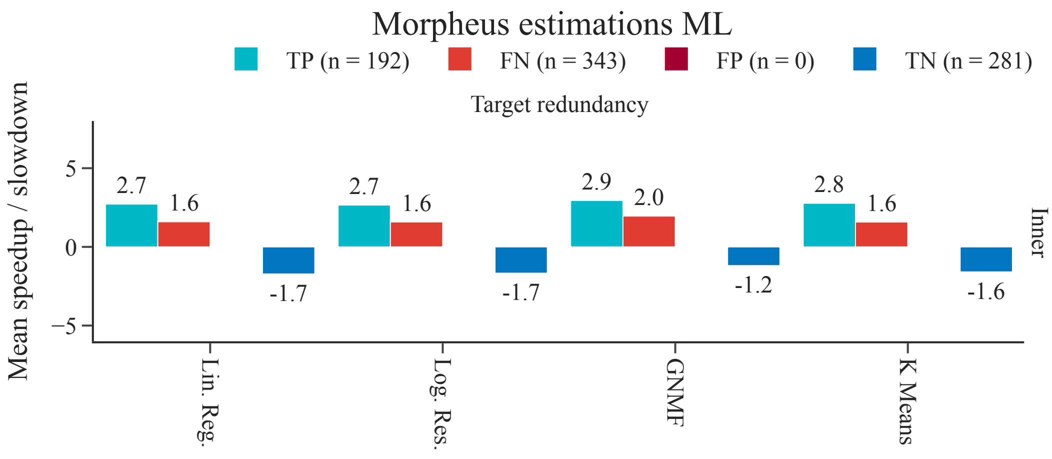


FIGURE 7.13: Mean speedups and slowdowns on ML models for correctly and incorrectly predicted cases by Morpheus' cost estimation on synthetic datasets.

We inspect the performance of Amalur's cost estimation in Figure 7.14. Here we also see that **the mean speedup for true positives is higher than the mean speedup for false negatives for inner joins and left joins, while for outer joins we see the opposite**. Overall, **the mean speedup for true positives is higher than the mean speedup for false negatives**. For left joins, the difference in mean speedup between true positives and false negatives is larger than for the inner joins. Our cost estimations, like Morpheus', do not include false positives ($n = 0$), but do include a higher number of false negatives ($n = 1108$) compared to the number of true negatives ($n = 940$). Again, the number of true positives ($n = 816$) equals the number of cases where speedups occur.

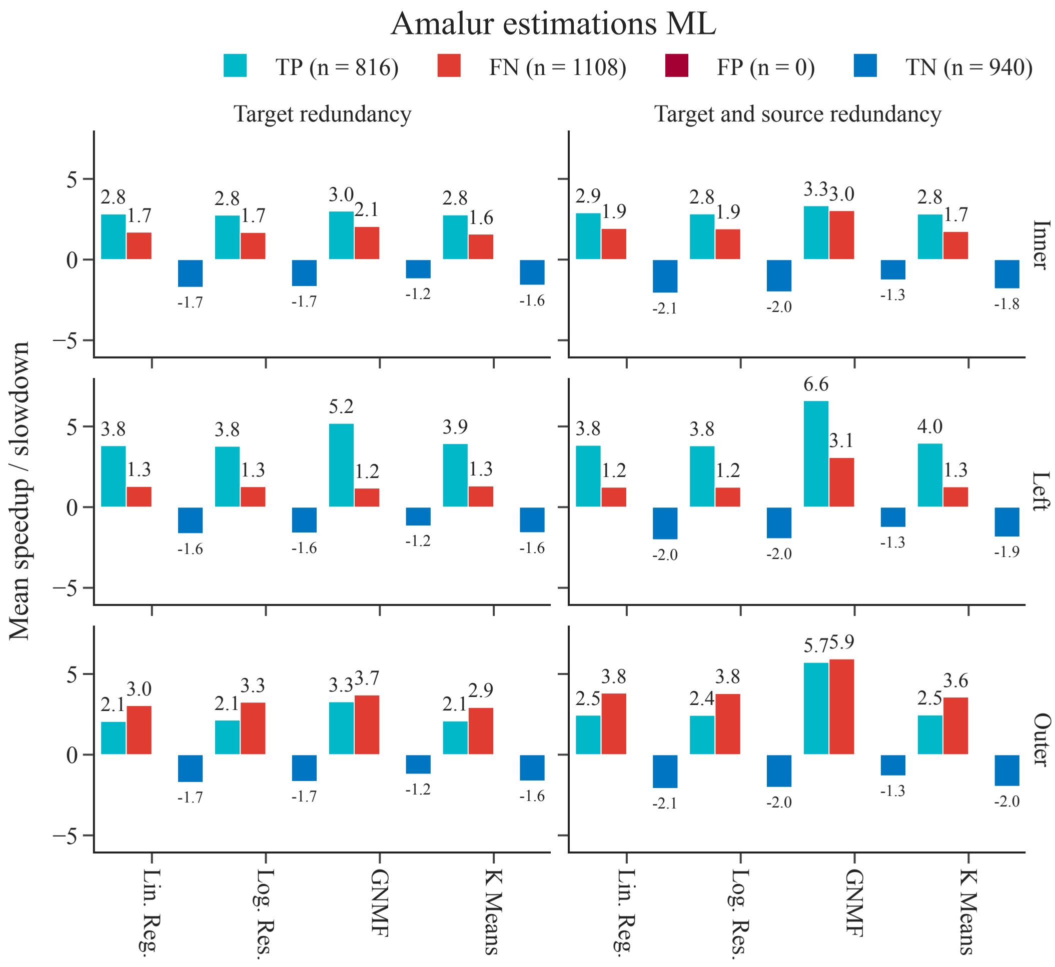


FIGURE 7.14: Mean speedups and slowdowns on ML models for correctly and incorrectly predicted cases by our cost estimation on synthetic datasets.

7.4.4 Overhead of cost estimation

Cost estimation involves obtaining parameters describing the size and sparsity of each source table S_k, of the target table T and the size of intermediate results. These parameters can be directly obtained from the normalized matrix, except the sparsity of T. This sparsity can be calculated after performing materialization. In order to determine the overhead of our cost estimation approach, we inspect the time it takes to perform the materialization operation described in Section 4.4.6. Figure 7.3 shows the cost estimation overhead relative to the time it takes to perform 20 iterations of each ML model Here we see that, **compared to only a few iterations of an ML model, the materialization time is negligible**. Considering that 20 iterations is a relatively low number of iterations compared to the number of iterations that are usually required for a ML model to converge, we conclude that **the overhead of our cost estimation approach is low enough for it to be included in Amalur without issues**.

7.5 Discussion

In this chapter, we first evaluated the performance of factorized learning in Amalur. Experiments on real datasets show that speedups occur for most ML models on most of the real datasets included in our experiments. Experiments on synthetic datasets show that we can achieve higher speedups for higher complexity ratios. In addition to indicating the performance of Amalur, this result also confirms that a cost estimation method based on the complexity ratio should perform well.

To fairly compare our cost estimation method to Morpheus' cost estimation method, we first calibrated Morpheus' method. Then, we compared the two methods on both real and synthetic datasets. We evaluated Morpheus' method on scenarios for which it was designed. Our method shows comparable performance to Morpheus' method in these scenarios on real datasets and synthetic datasets. For small datasets, Morpheus' performance would be much lower if we had not added an additional rule during our calibration. Our method performs well on scenarios both with target redundancy and scenarios with target redundancy and source redundancy and can provide predictions for multiple types of joins. Overall, both our cost estimation as well as Morpheus' cost estimation are more likely to mispredict small speedups than large speedups, and are more likely to mispredict small slowdowns than large slowdowns.

Our evaluation is done on relatively small datasets. While this can be seen as a limitation since factorization is more relevant to larger datasets with long model training times, it has allowed us to investigate the effectiveness of cost estimation approaches, as there are more cases without significant speedups for smaller datasets.

Contributions discussed in this chapter

– The evaluation of factorized learning in Amalur compared to learning over materialized data, which shows that we achieve speedups for some datasets and not for others,
– The evaluation of our cost estimation procedure compared to Morpheus' cost estimation procedure. This evaluation shows that we often outperform Morpheus' cost estimation on scenarios for which it was designed, and that we cover previously uncovered data scenarios with similar performance.

Chapter 8

Conclusion

This thesis aims to accurately estimate in which DI scenarios factorization can help to speed up the training of ML models. We make four contributions towards solving this problem. First, we implement an efficient data representation and an API for ML factorization. This contribution **C1** is the system Amalur described in Chapter **C1**. It provides us with an implementation of factorized learning which supports the DI scenarios we are interested in evaluating. Second, we develop a data generator to generate user-specified data integration scenarios. This contribution, which is **C2**, creates the inputs to Amalur which we need to evaluate a diverse set of DI scenarios. This evaluation is our third contribution **C3**. Our last contribution, **C4**, is the design, implementation, and evaluation of a cost estimation procedure for Amalur. This contribution combines the first three contributions and relates to the primary goal of this thesis.

8.1 Research questions

After creating the implementation of Amalur, generating data integration scenarios, evaluating Amalur, and creating a cost model, we can answer our research questions.

> **RQ1**: **How can we create an efficient implementation of ML factorization in the system called Amalur?**

We implemented factorized learning in Amalur by defining rewrite rules for LA operations both in the standard and transposed case and implementing four ML models using these operations. Extending our system with new ML models consisting of these operations is straightforward. The most important contribution we made towards the efficiency of our implementation is the sparse matrix representations for source tables, indicator matrices, and mapping matrices. In addition, we provide an optimal ordering of intermediate computations involving these matrices. To further optimize Amalur, we provide optimizations for specific scenarios, which are scenarios without column overlap and star schema joins. Our evaluation shows that Amalur's factorized learning results in speedups for most datasets and ML models.

> **RQ2**: **How can we generate data integration scenarios on which to test systems for factorized ML?**

We create a data generator through which users can generate datasets that resemble real data integration scenarios. To create data integration scenarios on which to test factorized ML, the most crucial factor to influence is redundancy. Using our generator, a user can influence both target and source redundancy. We allow the user to select which type of join they want to generate and adapt input parameters to the join type. In addition to target redundancy and source redundancy, the user can influence sparsity and the number of source tables in the dataset.

> RQ3: **How can we effectively perform cost estimation in systems for factorized ML?**

The performance of factorization depends upon the amount of redundancy present in the target table compared to the source tables. Therefore, we define two types of redundancy: target redundancy and source redundancy. More target redundancy results in better performance of factorization. We can quantify the amount of redundancy by computing the number of arithmetic computations involved in the standard and in the factorized case.

> RQ3a: **Which variables affect the cost of performing ML on materialized data?**

The cost of learning ML models on materialized data depends on which LA operations are included in the model. The cost of performing LA operations on materialized data depends upon the total number of records in the materialized table and, for matrix multiplication, the number of records in the matrix we are multiplying with.

> RQ3b: **Which variables affect the cost of performing factorized ML?**

The cost of performing factorized ML also depends upon the LA operations included in the model. The cost of performing factorized LA operations depends upon the total number of records in each source table and, for matrix multiplication, the number of records in the matrix we are multiplying with.

8.2 Future work

In this section, we propose future work directions for factorized learning and cost estimation.

8.2.1 Factorized learning

One aspect of factorization not covered in this thesis is the **generation of indicator and mapping matrices**. We assume that these follow directly from entity resolution and schema matching DI techniques and therefore can easily be created by DI systems. In future work, Amalur should be integrated with such a system. Special attention should be given to categorical variables during such an integration, since these variables might cause additional difficulties when generating mapping matrices. When we apply one-hot-encodings to be able to run ML models on our data, one categorical column turns into multiple columns. The challenge is to keep the mapping matrices consistent with such a change. Special attention should also be given to **null values**. In this work, we turned any null values into zeroes. Future work might include other methods of handling null values.

While we obtain speedups using our implementation based on sparse matrices, future work is needed to optimize Amalur further. Firstly, the structure of the factorized rewrites can be exploited through **parallelization** [4]. By computing partial results for each source table concurrently, the computation times of aggregation and multiplication operations, which are prevalent in ML models, can be reduced. Our cost model can be adapted to reflect such an implementation. Second, our matrix ordering optimization only considers the dimensions of the matrices, while for sparse matrices, the sparsity of each of the matrices should be considered. This requires finding **an estimation for the numeric sparse matrix chain product ordering problem** [32]. Third, as previously identified by the authors of Morpheus[5], the main bottleneck in the implementation of ML factorization are **matrix-matrix multiplications**. Options to combat this bottleneck are solutions such as DeepMind's AlphaTensor[1], CuPy [2], or direct implementation in a more low-level programming language such as C++. Lastly, parts of the computations may benefit from a **caching mechanism**.

8.2.2 Cost estimation

While the difference between the number of arithmetic computations involved in factorized learning and learning over materialized data is essential to understanding when speedups occur, different hardware settings may also have a significant impact on factorization performance. Additional work is needed to **research the effect of hardware specifications on factorization performance**. In addition, our complexity analysis does not include the overhead caused by multiplication with mapping and indicator matrices, or the time it takes to perform the operations in ML models that do not involve the normalized matrix. By **incorporating these aspects into the cost estimation procedure**, it might be possible to obtain a more accurate estimation.

[1]AlphaTensor: `https://github.com/deepmind/alphatensor`
[2]CuPy: `https://cupy.dev/`

Appendix A

Dataset Schemas

This chapter details the schemas of the datasets in the Hamlet dataset, which were defined in [26].

Walmart The walmart data set is used to predict department-wise sales based on store data, weather data and economic data. The schema of table S_1 is Sales(SalesLevel, IndicatorID, StoreID, Dept). This table contains the dependent variable, SalesLevel. Table S_2 is called Indicators and contains weather data and economic data. The schema of S_2 is Indicators(IndicatorID, TempAvg, TempStdev, UnempRateAvg, UnempRateStdev, IsHoliday). Table S_3 stores store data, it has the schema Stores(StoreID, Type, Size).

Expedia This dataset can be used to predict the rank of a hotel in a search result based on search data and hotel data. Table S_1 is Listings(Position, HotelID, SearchID, Score1, Score2, LogHistoricalPrice, PriceUSD, PromoFlag, OrigDestDistance). The attribute Position is the rank of the hotel in the search result. Table S_2 is Hotels (HotelID, Country, Stars, ReviewScore, BookingUSDAvg, BookingUSDStdev, BookingCount, BrandBool, ClickCount). Table S_3 Searches (SearchID, Year, Month, WeekOfYear, TimeOfDay, VisitorCountry, SearchDest, LengthOfStay, ChildrenCount, AdultsCount, RoomCount, SiteID, BookingWindow, SatNightBool, RandomBool).

Flights Using this data we predict whether a flight route is codeshared based on route data, airline data and data about the source and destination airports. Table S_1 is Routes (CodeShare, AirlineID, SrcAirportID, DestAirportID, Equipment1, ..., Equipment20). The independent variable we are interested in is CodeShare. Table S_2 is Airlines (AirlineID, AirCountry, Active, NameWords, NameHasAir, NameHasAirlines). Table S_3 is SrcAirports (SrcAirportID, SrcCity, SrcCountry, SrcDST, SrcTimeZone, SrcLongitude, SrcLatitude). R3 is DestAirports (DestAirportID, DestCity, DestCountry, DestTimeZone, DestDST, DestLongitude, DestLatitude).

Yelp Using the Yelp dataset we predict how a user will rate a business based on data describing the business and data describing the user. Table S_1 is Ratings (Stars, UserID, BusinessID), where Stars is the rating we are aiming to predict. Table S_2 is Businesses (BusinessID, BusinessStars, BusinessReviewCount, Latitude, Longitude, City, State, WeekdayCheckins1, ..., WeekdayCheckins5, WeekendCheckins1, ..., WeekendCheckins5, Category1, 15, IsOpen). Table S_3 is Users (UserID, Gender, UserStars, UserReviewCount, VotesUseful, VotesFunny, VotesCool).

MovieLens1M The Movies dataset predicts how a user will rate a movie based on data describing the user and data describing the movie. Table S_1 is Ratings (Stars, UserID, MovieID). The rating we are aiming to predict is stored in column Stars. Table S_2 is Movies(MovieID, NameWords, NameHasParentheses, Year, Genre1, . . . , Genre18). Table R_2 is Users (UserID, Gender, Age, Zipcode, Occupation). Both foreign keys have closed domains with respect to the prediction task.

LastFM The LastFM dataset can be used to predict a user's music play level based on data about past play levels, data about users and data about artists. Table S_1 is Plays (PlayLevel, UserID, ArtistID). The variable we are trying to predict is PlayLevel. Table S_2 is Artists (ArtistID, Listens, Scrobbles, Genre1, . . . , Genre5). Table S_3 is Users (UserID, Gender, Age, Country, JoinYear).

BookCrossing Using the Books dataset we predict how a user will rate a book based on data about users and data about books. Table S_1 is Ratings (Stars, UserID, BookID). The rating we are aiming to predict is stored in column Stars. Table S_2 is Users (UserID, Age, Country). Table S_3 is Books (BookID, Year, Publisher, NumTitleWords, NumAuthorWords).

Appendix B

Runtime plots on transposed data

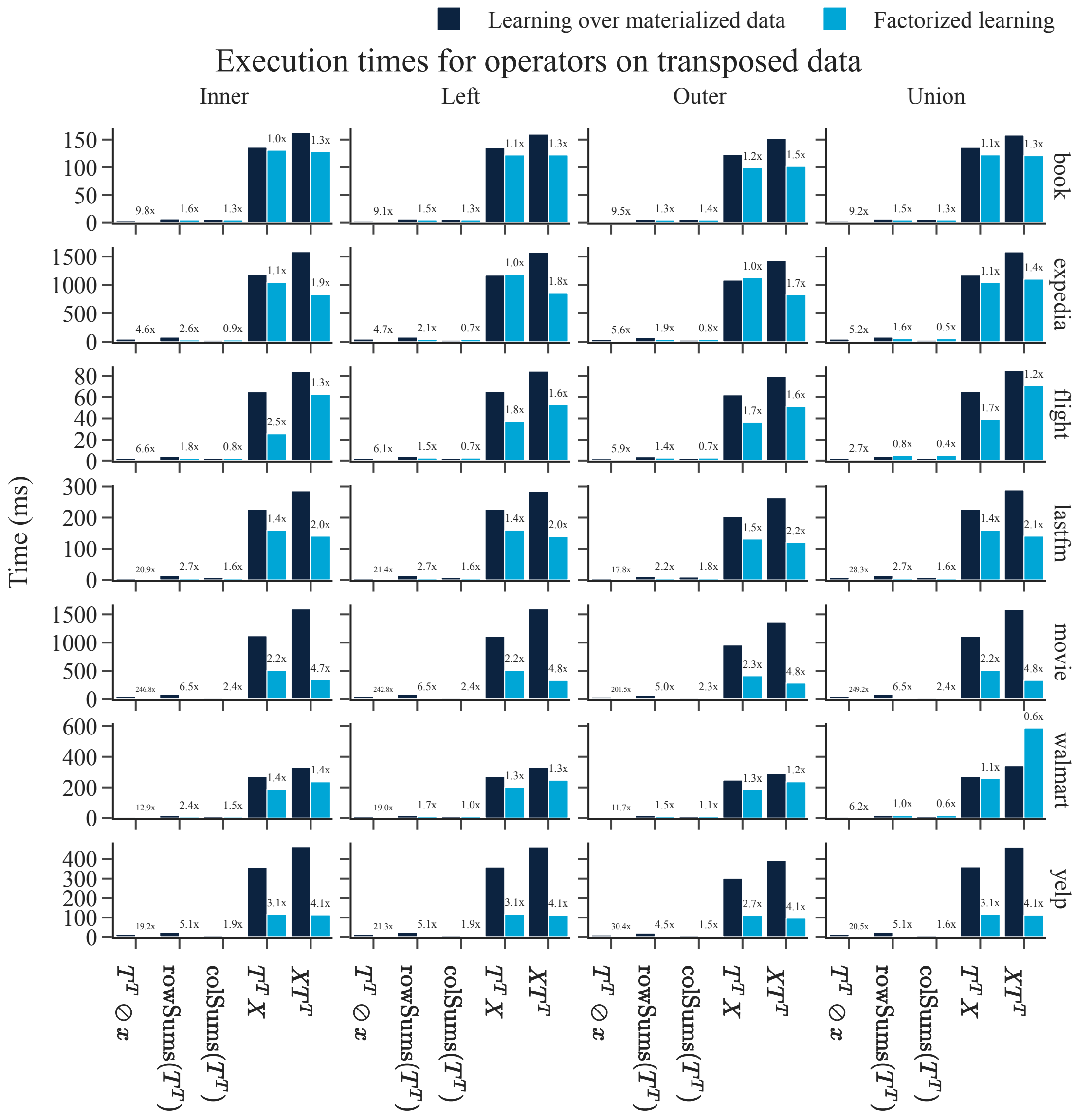


FIGURE B.1: Evaluation of Amalur LA operators on Hamlet dataset for transposed data.

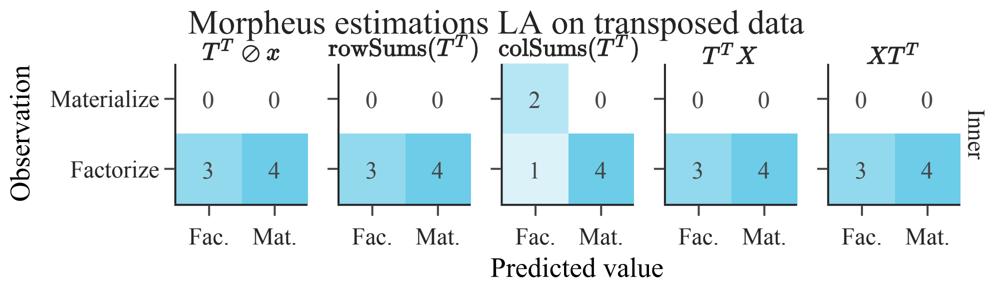


FIGURE B.2: Confusion matrices for Morpheus cost estimation on LA operators for transposed data. Each confusion matrix is annotated with the precision p.

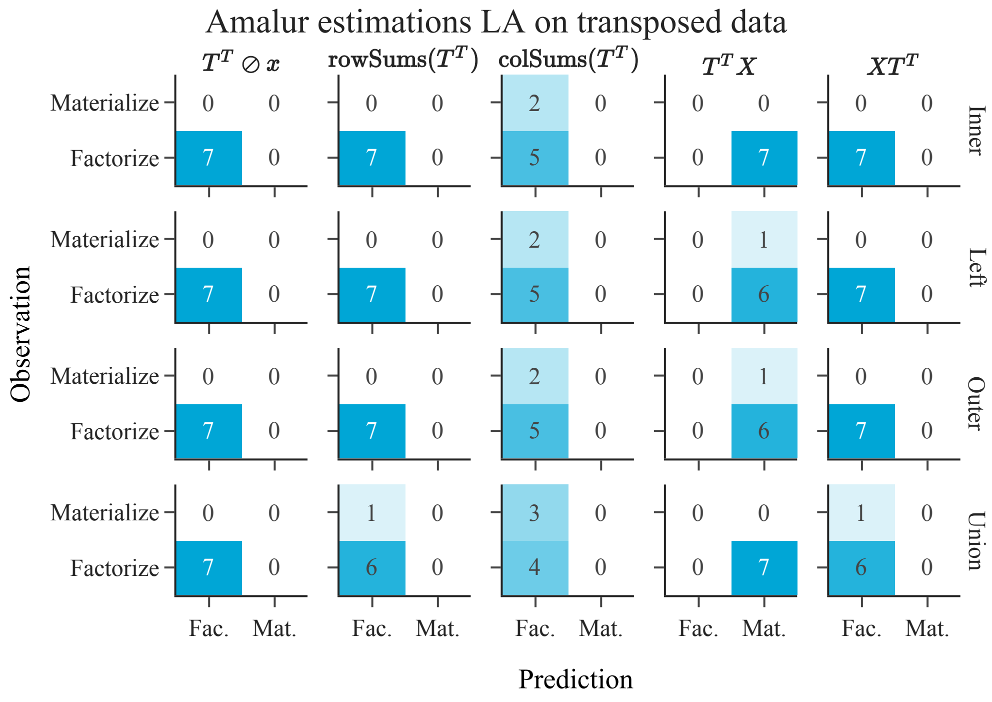


FIGURE B.3: Confusion matrices for Amalur cost estimation on LA operators for transposed data. Each confusion matrix is annotated with the precision p.

Bibliography

[1] R. Alotaibi, B. Cautis, A. Deutsch, and I. Manolescu, "HADAD: A Lightweight Approach for Optimizing Hybrid Complex Analytics Queries," en, in *Proceedings of the 2021 International Conference on Management of Data*, Virtual Event China: ACM, Jun. 2021, pp. 23–35.

[2] P. C. Arocena, R. Ciucanu, B. Glavic, and R. J. Miller, "Gain control over your integration evaluations," en, *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1960–1963, Aug. 2015.

[3] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller, "The iBench integration metadata generator," en, *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 108–119, Nov. 2015.

[4] A. R. Benson and G. Ballard, "A framework for practical parallel fast matrix multiplication," en, in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Francisco CA USA: ACM, Jan. 2015, pp. 42–53.

[5] L. Chen, A. Kumar, J. Naughton, and J. M. Patel, "Towards linear algebra over normalized data," en, *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1214–1225, Aug. 2017.

[6] Z. Cheng and N. Koudas, "Nonlinear Models Over Normalized Data," in *IEEE International Conference on Data Engineering (ICDE)*, Apr. 2019, pp. 1574–1577.

[7] Z. Cheng, N. Koudas, Z. Zhang, and X. Yu, "Efficient Construction of Nonlinear Models over Normalized Data," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, Apr. 2021, pp. 1140–1151.

[8] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis, *End-to-End Entity Resolution for Big Data: A Survey*, arXiv:1905.06397 [cs], Aug. 2020.

[9] E. F. Codd, "A relational model of data for large shared data banks," en, *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[10] E. F. Codd, "Relational database: A practical foundation for productivity," in *ACM Turing award lectures*, New York, NY, USA: Association for Computing Machinery, Jan. 2007, p. 1981.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, fourth edition*, en. MIT Press, Apr. 2022.

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

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

[14] A. Doan, A. Halevy, and Z. Ives, *Principles of Data Integration*, en. Elsevier, Jun. 2012.

[15] X. L. Dong and T. Rekatsinas, "Data Integration and Machine Learning: A Natural Synergy," en, in *Proceedings of the 2018 International Conference on Management of Data*, Houston TX USA: ACM, May 2018, pp. 1645–1650.

[16] H. Garcia-Molina, J. D. Ullman, and J. Widom, "Database Systems: The Complete Book," en, Jan. 2002.

[17] B. Golshan, A. Halevy, G. Mihaila, and W.-C. Tan, "Data Integration: After the Teenage Years," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, May 2017, pp. 101–106.

[18] R. Hai, C. Koutras, A. Ionescu, and A. Katsifodimos, "Amalur: Next-generation Data Integration in Data Lakes," en, Sep. 2022.

[19] R. Hai, C. Quix, and D. Wang, "Relaxed Functional Dependency Discovery in Heterogeneous Data Lakes," en, in *Conceptual Modeling*, ser. Lecture Notes in Computer Science, Springer International Publishing, 2019, pp. 225–239.

[20] A. Halevy, F. Korn, N. F. Noy, *et al.*, "Goods: Organizing Google's Datasets," en, in *Proceedings of the 2016 International Conference on Management of Data*, ACM, Jun. 2016, pp. 795–806.

[21] A. Halevy, A. Rajaraman, and J. Ordille, "Data integration: The Teenage Years," in *Proceedings of the 32nd international conference on Very large data bases*, ser. VLDB '06, Sep. 2006, pp. 9–16.

[22] D. Justo, S. Yi, L. Stadler, N. Polikarpova, and A. Kumar, "Towards a polyglot framework for factorized ML," en, *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2918–2931, Jul. 2021.

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

[24] 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.

[25] A. Kumar, J. Naughton, and J. M. Patel, "Learning Generalized Linear Models Over Normalized Data," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, May 2015, pp. 1969–1984.

[26] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu, "To Join or Not to Join?: Thinking Twice about Joins before Feature Selection," en, in *Proceedings of the 2016 International Conference on Management of Data*, dat, Jun. 2016, pp. 19–34.

[27] S. L. A. Kumar, "MorpheusPy: Factorized Machine Learning with NumPy," en, p. 10,

[28] D. Lee and H. S. Seung, "Algorithms for Non-negative Matrix Factorization," in *Advances in Neural Information Processing Systems*, vol. 13, MIT Press, 2000.

[29] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "How good are query optimizers, really?" en, *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 204–215, Nov. 2015.

[30] M. Lenzerini, "Data integration: A theoretical perspective," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Jun. 2002, pp. 233–246.

[31] S. Li, L. Chen, and A. Kumar, “Enabling and Optimizing Non-linear Feature Interactions in Factorized Linear Algebra,” en, in *Proceedings of the 2019 International Conference on Management of Data*, ACM, Jun. 2019, pp. 1571–1588.

[32] U. Naumann, “On Sparse Matrix Chain Products,” in Jan. 2020, pp. 118–127.

[33] 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*, Jun. 2016, pp. 3–18.

[34] M. Schleich, D. Olteanu, M. A. Khamis, H. Q. Ngo, and X. Nguyen, “A Layered Aggregate Engine for Analytics Workloads,” en, *Proceedings of the 2019 International Conference on Management of Data*, pp. 1642–1659, Jun. 2019.

[35] V. Shah, A. Kumar, and X. Zhu, “Are Key-Foreign Key Joins Safe to Avoid when Learning High-Capacity Classifiers?” *arXiv:1704.00485 [cs]*, Jun. 2017.

[36] J. Sun, Z. Shang, G. Li, D. Deng, and Z. Bao, “Dima: A distributed in-memory similarity-based query processing system,” *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1925–1928, Aug. 2017.

[37] Z. Zhao, R. Christensen, F. Li, X. Hu, and K. Yi, “Random Sampling over Joins Revisited,” in *Proceedings of the 2018 International Conference on Management of Data*, Association for Computing Machinery, May 2018, pp. 1525–1539.