IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 8 September 2025, accepted 13 October 2025, date of publication 17 October 2025, date of current version 23 October 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3622752

RESEARCH ARTICLE

Lightweight Semantic Similarity Search of Data Products

ANTONIO D'AMBROSIO[1], PAOLO PLATTER[1], MARTINA SALIS[2], FRANCESCO SIMBOLA[2], DIEGO REFORGIATO RECUPERO[2], AND DANIELE RIBONI[2]

[1]AgileLab S.r.l., 20121 Milan, Italy
[2]Department of Mathematics and Computer Science, University of Cagliari, 09124 Cagliari, Italy

Corresponding author: Martina Salis (martina.salis@unica.it)

This work was supported in part by the Data Mesh Platform Builder with AI (DAMPAI), through Fondo per la crescita sostenibile by the Ministero delle Imprese e del Made in Italy; and in part by the University of Cagliari "Open Access Awards: Call for Applications for Contributions to the Open Access Publication" under Grant DR 344/2025.

ABSTRACT The increasing popularity of digital services provided by large enterprises has determined a proliferation of data products, determining several data management issues. Among them, the recognition of overlapping data products is a major challenge, since data product duplication can lead to inefficiencies, unnecessary overhead, and customer confusion. Previous works have proposed different methods to search for joinability or semantic similarity in relational database systems. Other recent works explored the use of Large Language Models (LLMs) for semantic similarity search in Big Data repositories. In this work, we propose a novel, lightweight technique to find possible duplications in data product catalogs, exploiting pre-trained transformer models and the Hungarian algorithm. With respect to previous works, our method does not incur the high computational overhead of LLMs, and is flexible enough to also detect partial duplication. Moreover, being based on metadata analysis, our system is applicable also to prospective or new data products that have not yet been instantiated with data. We conducted an experimental evaluation with a set of data products, measuring the accuracy of similarity detection against a gold standard and the computational cost. Experimental results support the effectiveness and efficiency of our solution.

INDEX TERMS Data product duplication, data mesh, semantic similarity, transformer models, metadata analysis.

I. INTRODUCTION

As organizations adopt the Data Mesh paradigm, shifting data ownership to domain teams and enabling them to design and manage their own Data Products, they often see a rapid increase in the number of products created across the enterprise [1], [2]. While this decentralization promotes innovation and scalability, it also presents a critical challenge: ensuring that each new Data Product is unique, relevant, and not redundant [3].

In the absence of effective governance and coordination, teams may unintentionally develop overlapping or duplicate Data Products, for example, by exposing datasets that are already available in the catalog. This can lead to confusion, inefficiencies, and inconsistent analytical results [3]. Consider a large retail company where both the marketing and supply chain teams independently publish a Data Product titled Top-Selling Products. One defines ''top-selling'' by total revenue, the other by units sold, producing conflicting outputs and eroding trust in the data ecosystem.

Such duplication introduces multiple problems: redundant work, higher maintenance overhead, consumer confusion, and challenges in enforcing data governance standards. Detecting these overlaps manually is not only time-consuming, but also prone to error, especially in large, distributed organizations where teams may lack visibility into each other's data assets.

To address this challenge, recent research has proposed a variety of technical and organizational approaches. These include schema- and metadata-based similarity detection techniques to identify redundant or semantically overlapping Data Products [1], [3], [4]. Graph-based representations of

The associate editor coordinating the review of this manuscript and approving it for publication was Chang Choi.

data assets combined with similarity scoring algorithms, such as cosine similarity, have been explored to automate the identification of potential duplications [5]. More recently, Large Language Models (LLMs) have been employed to enhance the semantic understanding of Data Product descriptions, enabling more effective matching even when naming conventions and schema structures differ significantly [6]. LLMs can also support automated catalog enrichment, metadata harmonization, and natural language querying, further reducing the risk of unintentional redundancy.

While LLMs offer powerful capabilities for semantic similarity and metadata enrichment, they often come with high computational costs and deployment complexity.

To address the problem of Data Product duplication in a more lightweight and practical manner, we developed the Data Product Similarity system, a streamlined solution that can run efficiently on any standard server infrastructure and delivers results significantly faster than LLM-based alternatives. This system automatically detects and flags potential duplication by comparing newly created Data Products with existing ones, assisting development teams in evaluating whether the data they intend to publish already exists within the organization in a similar form. By leveraging schema-level structural analysis rather than deep semantic embeddings, the solution ensures a good balance between performance, simplicity, and operational feasibility.

The system calculates a similarity score between the input Data Product and a set of existing Data Products stored in the database by analyzing and comparing their schemas. The score ranges from 0 to 1, where values closer to 1 indicate a higher degree of structural similarity and a greater likelihood of duplication. To determine the optimal alignment between the schema elements of two Data Products, the system applies the Hungarian algorithm [7]. This algorithm operates on a cost matrix, where each element represents the aggregated distance, computed from both the name and description, between a pair of schema items. By minimizing the total cost of this matrix, the algorithm identifies the most efficient one-to-one matching between elements.

By surfacing these insights early in the development lifecycle, the system enables teams to make more informed decisions, avoid redundant work, and promote the reuse of existing data assets, ultimately contributing to a more efficient and coherent data mesh ecosystem.

More in detail, the specific contributions of our work are the following:

- the development of a novel approach based on the Hungarian algorithm to minimize the total cost in the cost matrix derived from item distances in Data Products;
- a significant reduction in processing time on the same set of Data Products compared to the time required to generate the gold standard outputs using the LLM-based approach;
- the construction of a gold standard through the use of an LLM;
- the experimental evaluation and comparison between the LLM-generated gold standard and the results produced by the proposed algorithm;
- a detailed analysis of the errors and misclassifications made by the proposed algorithm with respect to the gold standard.

The remainder of this manuscript is organized as follows. Section II reviews related work on similarity detection in data mesh systems. Section III provides a definition of data products and introduces the Hungarian algorithm used in our approach. The architecture of the proposed system is described in Section IV, while Section V outlines the methodology adopted for constructing the gold standard. Section VI presents the experimental evaluation, and Section VII discusses the corresponding error analysis. The computation analysis is also reported in Section VIII. Finally, Section IX concludes the paper and outlines directions for future work.

II. RELATED WORK

A data product is a digital product or service leveraging data to provide some utility to its consumers. It is typically accompanied by interfaces for data query and manipulation, and by authorization mechanisms to determine legitimate access. Data products can provide their services in different forms, including structured, semi-structured, or unstructured data. In order to enhance interoperability, data products typically expose formal schemas; i.e., metadata describing their structure and semantics. More details about data products are given in Section III-A.

The increasing popularity of the data mesh paradigm [2] has determined a proliferation of data products, leading to different data management challenges, including the detection of duplicated data products. Indeed, especially in data lakes of large organizations, different departments or units can inadvertently store the same data in different forms [3], compromising the efficiency of the infrastructure and possibly offering redundant data products. As a consequence, in the following, we will reference different research studies that addressed the problem of detecting similarity or duplication in big data and data lake systems.

Several methods have been proposed to find similarity between fields of different data structures in big data systems [8]. This problem is commonly referred to as *joinability discovery*, since the goal of many of these systems is to find tables that can be joined together based on latent foreign key constraints among their columns. In this research field, cell-level methods, like the one proposed by Zhu et al. [9], measure joinability by counting the exact matches among the cells of distinct columns, possibly relying on embedding technologies to better capture semantic matching, as proposed by Dong et al. [10]. Cell-based methods are very effective but are characterized by high computational overhead. Instead, column-level methods generally rely on computing a single representative vector to encode the whole column for the sake of similarity search. Dong et al. [11]

propose to use pre-trained transformer models to represent the whole column content through a single vector embedding. Instead, Flores et al. [12] propose to consider both columns similarity, measured through Levenshtein distance between several features extracted from column values, and instance cardinality proportions. With respect to cell-level methods, column-level ones provide approximate results, but aim to enhance scalability. Other approaches consider not only the data instances, but also metadata for semantic matching. In particular, Bharadwaj et al. [13] propose to use a machine learning-based approach. They extract structured data features both from column metadata and from samples of column instances, and train a random forest classifier to decide whether two columns are related or not. However, with respect to the objective of our work, that is, finding similar data products, the fact that two data structures are joinable does not necessarily mean that they convey similar content.

Other works address the problem of finding similarity between entire data structures. The R2D2 system [5] adopts a three-step hierarchical pipeline for detecting dataset containment in data mesh systems. The pipeline first builds a schema containment graph followed by statistical min-max pruning and content-level pruning. Detected redundancies are solved by a latency-aware mechanisms for reconstructing duplicated data on demand. However, in that work, duplication is detected only when the contents of a dataset are exactly contained within another dataset. In our work, we pursue a more flexible approach, aiming at detecting duplication even when there is substantial (but not necessarily complete) overlapping between two data products.

Bogatu et al. [14] propose a method for similarity search in data lakes extracting features from both schema and instances of a target data product. They use different features, including attribute name similarity, attribute value similarity, attributes word-embedding similarity, as well as numerical and string pattern similarity among instance values. Extracted features are mapped to a common feature space for distance-based retrieval. Though effective, that approach is computationally expensive, since it considers not only metadata but also data instances, which in big data systems may incur severe computational overhead. Moreover, in that work, indexes must be globally recomputed at each new dataset insertion. On the contrary, our approach is lightweight, since it considers only the product metadata, and does not require global parameter updates when additional data products are considered.

A hierarchical approach to data similarity search is proposed by Ouellette et al. [15]. Each structured dataset is represented by averaging the word embedding vectors of its schema and metadata tokens. The similarity between two data structures is computed by the cosine similarity of the corresponding vectors. As a consequence, in that work, the structure of the metadata is not considered, while in our work we leverage structural information that may be required to properly capture the data semantics. Other recent works exploit LLM technologies for similarity detection in data mesh systems [16]. However, the use of LLM technologies incurs high computational overhead; hence, in our work, we present a lightweight system that can operate efficiently in large-scale environments.

Finally, we mention that other works address the problem of *unionable tuple search*; i.e., finding different data structures that share similar schema and instances, typically with the goal of data integration. In this field, similarity is generally assessed through tables' columns overlap [17], semantic similarity among data instances [18], or metadata similarity [19]. However, in this paper, we do not pursue the goal of data integration. Instead, we address the problem of detecting redundancies in large data product catalogs, in order to avoid inefficiencies in storage, management and retrieval. For this reason, our technique aims at finding pairs of redundant data products, even though they are not necessarily suitable for merging. Moreover, since we rely only on metadata information, our method is applicable also to new or prospective data products that have not yet been instantiated with content.

III. BACKGROUND

In this section, we provide background information on data products and the Hungarian algorithm, which we use in our proposed method to compute the semantic similarity between them.

A. WHAT IS A DATA PRODUCT

A *Data Product*[1] is a reusable and self-contained unit of data and metadata, designed to deliver value to consumers through standardized and discoverable interfaces. The concept originates from the data mesh paradigm [2] and has been further refined in [20]. Embracing data mesh principles, data products represent a shift from traditional pipeline-centric architectures toward a product-oriented mindset, where data is treated as a first-class asset. Each data product is domain-owned, trustworthy, addressable, and built for consumption by other domains or systems within a decentralized data architecture.

Typically, a data product includes the following core components:

- **Metadata:** Descriptive fields such as `id`, `name`, `kind`, `environment`, `domain`, and `description`.
- **Components:** Modular sub-elements such as input ports, output ports, and processing logic.
- **Data Contracts:** Schema definitions associated with components, specifying field names, data types, and semantic descriptions.
- **Interfaces:** Mechanisms for exposing or consuming data, often via APIs or message brokers.
- **Ownership and Governance:** Domain affiliation and potentially access control definitions.

Specifically, an *output port* specifies how a data product makes its data available to consumers, and saying that it

[1] https://www.datamesh-manager.com/learn/what-is-a-data-product

```yaml
id: urn:dmb:cmp:customercare:customer-onboarding:0
kind: dataproduct
environment: development
name: customer onboarding
description: customer onboarding
domain: customercare
components:
- id: urn:dmb:cmp:customercare:customer-onboarding:0:customer-onboarding-optimization
  kind: outputport
  dataContract:
    schema:
    - dataType: INT
      description: Unique identifier for the customer
      name: customerId
    - dataType: STRING
      description: Current step in the customer onboarding process
      name: onboardingStep
    - dataType: DOUBLE
      description: Time taken to complete the onboarding step (in minutes)
      name: stepDuration
    - dataType: STRING
      description: Indicates if the onboarding step was completed successfully
      name: stepCompletionFlag
```

FIGURE 1. Example of data product.

exposes a schema means that it clearly defines the expected format of the delivered data, including the names, types, and meaning of each item, ensuring that consumers can correctly interpret and use it.

In addition, the descriptor of a data product includes a *schema*, which specifies the logical structure of the data. A schema defines the set of items, their data types, and semantic meaning, serving as a contract between the data product and its consumers to guarantee consistency and interoperability.

Figure 1 illustrates an example of a data product's descriptor with the identifier `urn:dmb:cmp:customercare:customer-onboarding:0`, defined in the `development` environment, belonging to the `customercare` domain and labeled `customer-onboarding-optimization`. This data product defines a single output port (listed under the `components` key), which exposes a well-defined schema. Each element of the schema might include different fields (i.e., type, name, description, etc.). We refer to each element as an item of the data product, and to its fields as the item's name or description. For example, the first item of the schema of the example is related to the identifier of a customer and includes three fields (dataType, description, and name). In this paper, we will consider Data Products whose elements of the schema have a name and a description.

The output port is identified by the ID `urn:dmb:cmp:customercare:customer-onboarding:0:customer-onboarding-optimization` and provides a schema with keys such as:

- `customerId (INT)`: Unique identifier for the customer,
- `onboardingStep (STRING)`: Current step in the customer onboarding process,
- `stepDuration (STRING)`: Time taken to complete the onboarding step (in minutes),
- `stepCompletionFlag (STRING)`: Indicates if the onboarding step was completed successfully.

This structured design ensures traceability, supports schema evolution, and enables semantic clarity and seamless consumption by both analytical and operational systems.

B. THE HUNGARIAN ALGORITHM

The *Hungarian algorithm*, also known as the *Kuhn-Munkres algorithm*, is a classical combinatorial optimization technique designed to solve the *assignment problem* in polynomial time. Introduced by Harold Kuhn in 1955 [7] and based on earlier mathematical work by Kőnig and Egerváry [21], [22], the algorithm identifies the optimal one-to-one matching between two sets of elements in a weighted bipartite graph, either minimizing total cost or maximizing overall similarity.

The algorithm operates on a square cost matrix C, where each element $C_{i,j}$ represents the cost (or inverse similarity) of assigning item i from one set to item j from another. The goal is to determine the assignment of elements that results in the lowest possible total cost (or highest similarity, by negating the matrix). The core steps of the algorithm are as follows:

1) **Row Reduction:** For each row, subtract the minimum value in that row from every element of the row.
2) **Column Reduction:** For each column, subtract the minimum value in that column from every element of the column.
3) **Cover All Zeros:** Use the minimum number of horizontal and vertical lines to cover all zeros in the matrix.
4) **Test for Optimality:** If the number of covering lines equals the number of rows (or columns), an optimal assignment exists.
5) **Adjust the Matrix:** If not optimal, subtract the minimum uncovered value from all uncovered elements, and add it to all elements covered twice. Repeat from Step 3.

The step-by-step transformation of the cost matrix during the execution of the Hungarian algorithm is illustrated in Figure 2, showing the key operations such as row and column reductions, zero-covering, and the identification of the optimal assignment.

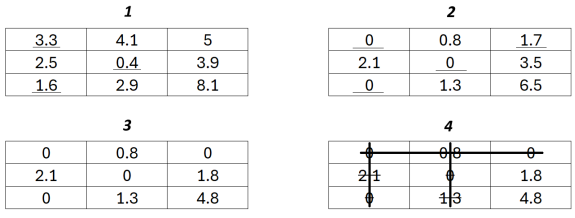


FIGURE 2. Step-by-step transformation of the cost matrix in the hungarian algorithm. From initial cost matrix (1), to row/column reductions (2, 3), and final optimal assignment (4).

In the context of data product comparison, consider two data products, each represented by a list of features or metadata descriptors. A similarity score, such as cosine similarity or Jaccard index, is computed between every possible pair of features from the two products, resulting in a similarity matrix.

To apply the Hungarian algorithm, the similarity matrix is first transformed into a cost matrix (e.g., by subtracting each similarity value from a fixed maximum). The algorithm

then computes the optimal one-to-one feature matching that maximizes total semantic similarity between the two data products. This matching allows for a quantifiable and interpretable measure of structural alignment between products, enabling downstream use cases such as metadata harmonization, duplication detection, or integration recommendation.

IV. ARCHITECTURE

This section presents the architecture underlying our approach. It outlines the core components of the system we implemented to identify similarity between Data Products. As shown in Figure 3, the block on the right encapsulates the various subcomponents that constitute the foundation of the entire system. The service can be described as a microservice that receives a Data Product in YAML format as input and compares it against previously processed Data Products stored in a database, as shown on the left-hand side of the figure.

The process begins with the input YAML structure being handled by the component responsible for schema analysis and normalization, referred to as the *Schema Analyzer* and discussed in Section IV-A. This is followed by the *Embedding Generation*, which is a transformation of textual content into vector representations, enabling meaningful semantic comparison (discussed in Section IV-B). The final component, *Distance Score Computation*, detailed in Section IV-C, computes the similarity between the input and stored Data Products by evaluating both names and schema descriptions for all pairs of data product schema items.

In the following sections, the terminology is used as follows: by *field* we refer to the attributes of each data product schema item (namely, the name or the description), whereas by *item* we refer to the data product schema element as a whole.

The Data Products stored in the database have already undergone preprocessing and embedding generation (the first two stages depicted in the architecture). As a result, their vector representations are precomputed and stored, allowing them to serve as a ready reference during the final similarity computation phase.

These three key components, *Schema Analyzer*, *Embedding Generation*, and *Distance Score Computation* form the backbone of the architecture and collectively enable accurate similarity detection. In the following, we will detail each of them.

A. SCHEMA ANALYZER

The *Schema Analyzer* is primarily responsible for the recursive preprocessing of the strings associated with the schema fields of a data product, specifically the column names and their descriptions. The *Schema Analyzer* is a modular component designed to improve the semantic clarity and consistency of textual metadata in a Data Product. It focuses specifically on the column names and column descriptions found within data schemas. These textual elements are often inconsistent, noisy, or ambiguous due to a lack of standardization across data sources. The analyzer applies a three-stage preprocessing pipeline to iteratively refine these strings and prepare them for downstream tasks such as schema matching, metadata enrichment, or semantic comparison.

This component includes four submodules, each applying a different approach to data preprocessing. The submodules are: *Regex-based Normalization*, *Tokenization with NTLK*, *Synonyms Matching*, and *Stop Word Removal*.

Each submodule is executed sequentially, in the order described, resulting in a progressively refined version of the input sentence. Each stage in the pipeline performs a specific function, building on the results of the previous one.

The first stage in the pipeline, *Regex-based Normalization*, is responsible for text cleaning through regular expression (regex) operations. Its primary goal is to remove noise and artifacts from the input string while preserving its original semantic meaning. This step ensures the input is in a standardized and readable format for the next stages of analysis.

The following transformations are applied in sequence:

1) *Removal of unwanted characters*: Special characters, redundant punctuation, and non-standard symbols are stripped from the string.
2) *CamelCase splitting*: Compound words written in camel case format (e.g., customerID) are split into separate tokens with spaces inserted (customer ID), which improves clarity and supports better tokenization.
3) *Letter-number boundary separation*: Transitions between letters and digits (e.g., amount100) or digits and letters (e.g., 2023Date) are detected and split by inserting spaces, resulting in more interpretable units.

This preprocessing step significantly reduces structural noise, resulting in a cleaner and more semantically consistent string ready for tokenization.

Once normalization is complete, the next step is *Tokenization with NLTK*; the process of splitting the cleaned string into discrete units called tokens (typically words). This step is implemented using the *Natural Language Toolkit (NLTK)* [23],[2] a powerful open-source library widely used for natural language processing (NLP) in Python. In this context, tokenization is critical because it enables fine-grained analysis of the schema fields (name and description) at the word level. By isolating individual terms, we make it easier to apply linguistic transformations, semantic matching, and comparisons between fields.

For example, given the phrase as input:

```
1) type of transaction 2 (e.g., credit,
        debit, creditCard)
```

applying the previously described operations yields the following final tokens:

[2]https://www.nltk.org/

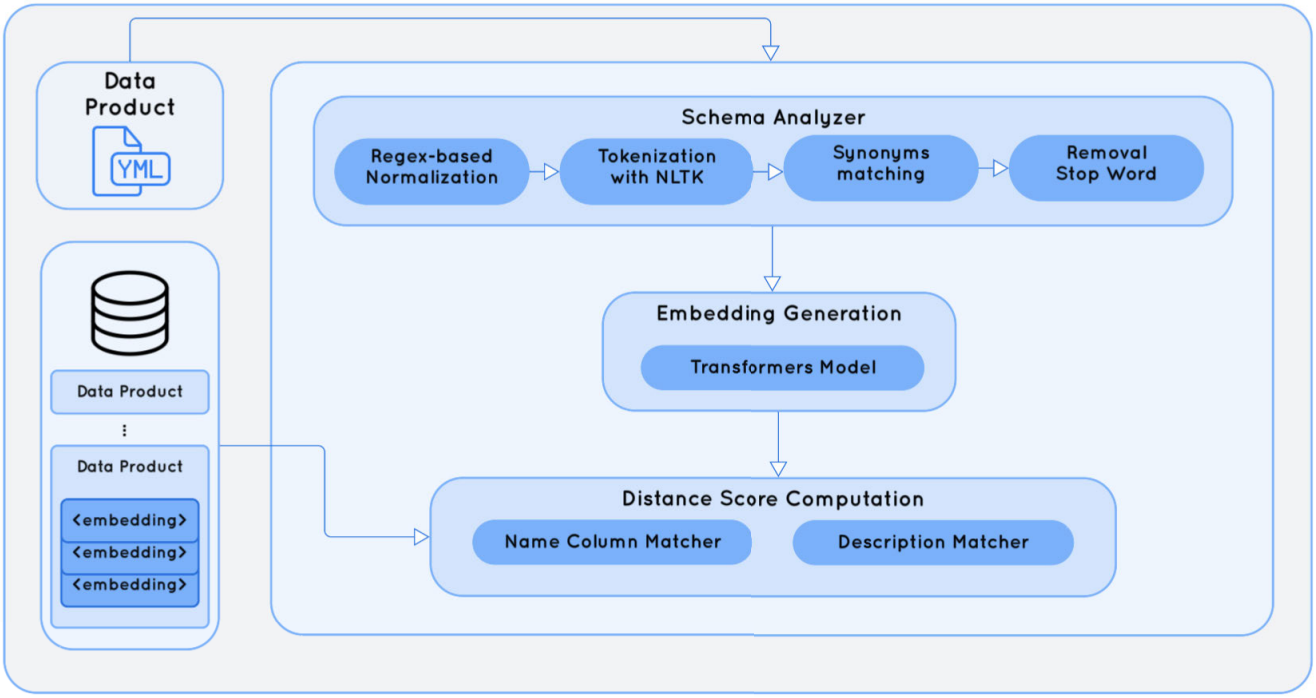


FIGURE 3. The architecture handles both the management of data products and the computation of duplication scores as part of its core functionality.

```
['1', 'type', 'of', 'transaction', '2',
 'e', 'g', 'credit', 'debit', 'credit',
                    'card']
```

The next step in the pipeline, *Synonyms Matching*, is focused on semantic normalization through synonyms mapping. For this purpose, a custom-built synonym dictionary is used, implemented as a `JSON` file. The file acts as a simple Python dictionary, where each key represents a canonical term, and each value is a list of synonyms associated with that term. For example:

```
{
    customer: [client, user],
    id: [key, identifier,
        ide, alias]
}
```

During processing, each token from the previous step is checked against the values in the synonyms dictionary. If a token matches any value in the list of synonyms, it is replaced with the corresponding key. For instance, the word "client" would be replaced with "customer".

This approach helps reduce linguistic variability and enhances semantic consistency across different schema fields. As a result, fields that use different terms for the same concept (e.g., client vs. customer) are normalized to a common vocabulary, improving the reliability of field comparison, clustering, and integration tasks.

The final step is the *Stop Word Removal* phase. Here an additional filtering step is applied to remove so-called "stop words", common words that typically carry little meaningful content in the semantic context, such as articles, prepositions, and conjunctions. This filtering relies on a predefined list of such words, sourced from a widely used and well-established linguistic library in the field of NLP. Removing these words helps focus on more relevant and distinctive terms, further improving the effectiveness of normalization and comparison operations. As a result of this final stage, we obtain a cleaner and more semantically consistent version of the input phrase. For example, the previously mentioned phrase is transformed into:

```
1 type transaction 2 credit debit credit
                    card
```

B. EMBEDDING GENERATION

The second module, *Embedding Generation*, processes input strings, that are the associated values of both fields name and description, using a Transformer-based architecture, optimized to generate dense vector representations that capture the semantic meaning of the text. The model, trained on large-scale text corpora, is designed for strong performance in semantic textual similarity tasks. Each input string is converted into a numerical vector that reflects its key semantic features. These vectors form the foundation for comparing data products in the similarity computation phase.

In this system, the pre-trained model `sentence transformers / all MiniLM-L6-v2` [24][3] is employed. This model, developed as part of the Sentence-Transformers framework, is a lightweight yet

[3]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

high-performing architecture comprising six Transformer layers. It has been fine-tuned on large-scale datasets using contrastive learning objectives to maximize its performance on semantic textual similarity (STS) tasks. For each input sentence, the model generates a dense vector of 384 dimensions, which encodes the core semantic features of the text in a high-dimensional vector space.

To ensure consistent and meaningful comparisons between embeddings, the output vectors are L2-normalized, effectively projecting them onto a unit hypersphere. This normalization is crucial when applying similarity metrics such as cosine similarity, allowing for the comparison of vector orientation regardless of their magnitude.

The resulting normalized embeddings serve as the foundation for the subsequent similarity computation phase, where the semantic proximity of different data products is assessed based on their vector representations.

C. DISTANCE SCORE COMPUTATION

The *Distance Score Computation* phase evaluates how closely related column names are to their corresponding descriptions by applying a similarity metric. Because column names are usually short and descriptions are more detailed, this metric helps effectively measure how well their meanings match.

In this phase, the system calculates the semantic distance between pairs of fields, specifically between the column names and their corresponding descriptions, using a suitable similarity metric. Typically, descriptions are longer and more detailed, while column names tend to be much shorter, often consisting of only one to three words. The chosen metric effectively captures the degree of relatedness between these two different types of textual content, enabling meaningful comparisons between concise identifiers (the field name) and their richer contextual descriptions.

The distance is computed separately for both the column names and the descriptions, which is why the architecture in the figure includes two distinct modules: the *Name Column Matcher* and the *Description Matcher*. Both modules follow the same computational procedure, leveraging identical similarity metrics and aggregation strategies; the only distinction lies in the type of input data: one operates on column names, while the other processes descriptions.

The following outlines the procedure carried out in both of the previously mentioned modules. For each input pair of the field of the two input data products schema, based on the configuration, either column names or descriptions, the system computes the distance using the corresponding precomputed embeddings. The pairs always refer to fields of the same type, meaning that the distance is calculated either between names or between descriptions of the two input data products.

The embeddings generated in the previous phase are compared using *cosine similarity*, a widely adopted metric in natural language processing for measuring the similarity between two vectors. Cosine similarity evaluates the cosine of the angle between two vectors in a multi-dimensional space, making it particularly suitable for comparing embedding vectors regardless of their magnitude. This allows the system to focus purely on the semantic orientation of the texts rather than their length or scale. Cosine similarity is defined as shown in equation (1):

$$\text{cosine\_similarity}(\mathbf{u}, \mathbf{v}) = \begin{cases} 0 & \text{if } \|\mathbf{u}\| = 0 \text{ or } \|\mathbf{v}\| = 0 \\ \dfrac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|} & \text{otherwise} \end{cases} \tag{1}$$

In the formula above:

- $\mathbf{u}$ and $\mathbf{v}$ represent the two vectors (embeddings) being compared.
- $\|\mathbf{u}\|$ and $\|\mathbf{v}\|$ denote the Euclidean norm (magnitude) of vectors $\mathbf{u}$ and $\mathbf{v}$ respectively, calculated as:

$$\|\mathbf{u}\| = \sqrt{\sum_{i=1}^{n} u_i^2}$$

  where n is the dimensionality of the vectors.
- $\mathbf{u} \cdot \mathbf{v}$ is the dot product (inner product) of the vectors, computed as:

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

After computing the cosine similarity between vectors, the distance is calculated by subtracting the similarity from 1. This distance measure is then clipped to the range [0, 1] to ensure numerical stability.

The distance is calculated for each pair of fields which consists of the name x and description y of the items in the two input data products schema. The distances of the name and description, d_x and d_y, are then aggregated into a single distance value, referred to as d_{xy}, for each pair of Data Product items, giving higher priority to the name than to the description during the summation process.

This priority is defined as a numeric weight configured in the system, where the name has a default value of 2 and the description has a default value of 1, thus giving the name higher priority over the description.

The distance d_{xy} is computed as a weighted sum of the field-level distances, giving higher weight to high-confidence matches and respecting the predefined field priority:

$$d_{xy} = \sum_{i=1}^{n} w_i \cdot d_i$$

where:

- n represents the number of fields, which in our case is equal to 2, as we only have the name and description fields,
- d_i is the distance between the i-th field of the two data products (which can be either the name or the description),

- w_i is the weight assigned to field i, based on its priority and its distance, $\sum_{i=1}^{n} w_i = 1$,

If the comparison between two fields (e.g., name) of the two input data products yields a distance $d_i < 0.2$, then a weight $w_i = 0.8$ is assigned to that field, and the remaining weight 0.2 is assigned to the other field's pair (e.g., description).

By default, priority is given to the name field. Therefore, if both the name and description fields have a distance less than or equal to 0.2, a weight of 0.8 is assigned to the name fields and 0.2 to the description fields. Conversely, if both field types (i.e., the distance between the name fields of the two items and that between the description fields) are greater than 0.2, each is assigned a weight of 0.5.

Example 1: Consider two items from two data products, with the distances between their fields labeled as x and y:

$$\text{distance between names: } d_x = 0.1$$
$$\text{distance between descriptions: } d_y = 0.6$$

Since $d_x \leq 0.2$, we assign:

$$w_x = 0.8, \quad w_y = 0.2$$

The aggregated distance d_{xy} is then:

$$d_{xy} = (0.1 \cdot 0.8) + (0.6 \cdot 0.2) = 0.08 + 0.12 = 0.20$$

Afterwards, a matrix m is constructed for the two input data products, where the rows and columns correspond to the schema items of each data product, respectively. Each element of this matrix represents the aggregated distance d_{xy} between the pair of items identified by the corresponding row and column.

The resulting distance matrix m serves as input to the Hungarian algorithm, which identifies a set of distance items, one per row and column, that minimizes the total matching cost.

The output of the Hungarian algorithm is therefore the optimal one-to-one matching between the schema items of the two data products, pairing each item from one data product with exactly one item from the other.

The optimal one-to-one matching between the schema items of the two input data products p_1 and p_2 is determined by selecting pairs of items (i, j), where i and j denote the indices of items from p_1 and p_2 respectively, such that the sum of their aggregated distances $d_{xy}(i, j)$ is minimized across all matched pairs. This ensures that each item from one schema is matched with exactly one item from the other schema, resulting in the lowest possible total matching cost.

The pairs of items (i, j) selected by the Hungarian algorithm are then used to compute a final distance score D_{p_1,p_2} between the two data products p_1 and p_2. This score is calculated as the percentage of item pairs whose aggregated distance $d_{xy}(i, j)$ is below the threshold of 0.3, providing a comprehensive measure of how closely aligned the two Data Product schemas are.

Formally, D_{p_1,p_2} is defined as:

$$D_{p_1,p_2} = 1 - \frac{\left|\{d_{xy}(i,j) \mid d_{xy}(i,j) \leq \tau\}\right|}{n}$$

where:

- $d_{xy}(i, j)$ is the aggregated distance between the matched items i of p_1 and j of p_2,
- τ is the similarity threshold (e.g., $\tau = 0.3$),
- n is the total number of matched item pairs selected by the Hungarian algorithm,
- $\{d_{xy}(i, j) \mid d_{xy}(i, j) \leq \tau\}$ is the set of distances considered sufficiently similar.

Importantly, the value of n corresponds to the number of matched pairs selected by the Hungarian algorithm, which operates on a cost matrix and finds the optimal assignment that minimizes the total distance (i.e., cost). In this context n is the number of optimal one-to-one matches between items in the two schemas of p_1 and p_2.

Example 2: Suppose that the Hungarian algorithm produces 5 matched pairs ($n = 5$) having the following distances:

$$\{0.2,\ 0.5,\ 0.1,\ 0.35,\ 0.6\}$$

Assuming $\tau = 0.3$, the distances below the threshold are:

$$\{0.2,\ 0.1\} \quad \Rightarrow \quad \text{count} = 2$$

Then D_{p_1,p_2} is:

$$D_{p_1,p_2} = 1 - \frac{2}{5} = 0.6$$

This indicates that only 40% of the items were sufficiently similar (i.e., below the threshold), resulting in a larger overall distance.

Finally, we compute the similarity score S as: $1 - D_{p_1,p_2}$, where D_{p_1,p_2} is the aggregated distance between p_1 and p_2.

The resulting similarity score ranges from 0 to 1, where a value closer to 1 indicates a high degree of similarity between the schemas, while a value closer to 0 suggests they are significantly different.

V. CREATION OF GOLD STANDARD

To establish a reliable benchmark for the evaluation phase, we manually created a set of 25 YAML descriptors,[4] each representing a real-world data schema. These descriptors belong to the FinTech domain, with a focus on loans, investments, payments, bank accounts, and ticketing. Each descriptor contains, on average, 9 to 10 items, modeled as dictionaries with standardized keys such as `name` and `description`. Item names follow a consistent *snake_case* format,[5] and the descriptors exhibit structural uniformity across schemas.

Common attributes include values like `id`, `date`, `amount`, `status`, `currency`, and `type`, which appear

[4]Publicly available at https://github.com/francescosimbola/DataProduct

[5]A naming convention where spaces are replaced by underscores and all letters are written in lowercase.

with high frequency. Descriptions are concise, written in English, and provide short but clear explanations of each item's meaning. Semantic similarities are often present even between items with different names, enabling meaningful schema comparisons. The consistency, clarity, and domain-specific focus of these descriptors make them particularly suitable for similarity evaluation and data governance applications.

For compatibility with language models, the descriptors were parsed into JSON format and provided through the API as input to an LLM for the construction of the gold standard.

The Gold Standard was designed to be as accurate and informative as possible. Each descriptor in the Gold Standard includes not only the ID used for similarity comparison, but also several additional contextual items. This extended structure ensures that the reference similarity judgments are based on a comprehensive understanding of the descriptors.

In contrast, our embedding-based model performs similarity computations in an item-by-item manner, independently comparing isolated components of the descriptors. As a result, it lacks a holistic view and may fail to capture the full semantic context shared between descriptors.

On the other hand, the LLM used to generate the Gold Standard was provided with complete descriptor inputs, including all relevant items at once. This allows the LLM to leverage global context and semantic relationships between items, leading to more accurate and robust similarity assessments. As a consequence, the LLM's evaluations are significantly more precise and context-aware than those produced by models operating on partial, item-level representations.

Subsequently, pairwise comparisons were conducted for all $25 \times 24 = 600$ (since we compare each data product with all the others) combinations using the GPT-4o model via API. Each comparison followed a standardized prompt in which only the content of the descriptors and their respective identifiers were changed. An example of the real prompt used to compare `descriptor_24` against `descriptor_0` is shown below in Figure 4.

First, the two descriptors are defined and illustrated, each identified by the key `id`. Next, the task is formally defined, along with the similarity measure and the corresponding formula used to compute it. It is important to note that the similarity score between `descriptor_A` and `descriptor_B` may differ from that of `descriptor_B` and `descriptor_A`, as the similarity measure is directional. This asymmetry arises because the denominator in the formula corresponds to the number of items in the second descriptor.

For example, in the comparison between `descriptor_24` (which contains 6 columns or items) and `descriptor_0` (which contains 30 columns or items), 4 items were identified as similar. Thus:

- `sim(descriptor_24 → descriptor_0)` = $4/30 = 0.13$
- `sim(descriptor_0 → descriptor_24)` = $4/6 = 0.67$

Prompt:
The descriptor is a YAML file that contains various pieces of information. Each descriptor follows a schema defined by the field "`columns`", which consists of a list of items representing columns in a database table. Each item includes a "`name`" and an optional "`description`".
We have the following two descriptors, each identified by the field "`id`":

```
descriptors = [
    {'id': 'descriptor_24', 'columns
        ': [...]},
    {'id': 'descriptor_0', 'columns':
         [...]}
]
```

Task: Compute the similarity index between `descriptor_24` and `descriptor_0`.
Similarity Definition:
1) Two items (i.e., column names and descriptions) are considered similar if they have equivalent meanings within the database domain.
2) Example: "`txId`" is similar to "`audit.txId`" since both refer to a transaction identifier.
3) Example: "`sourceTable`" is similar to "`audit.sourceTable`" as they both refer to the source database table.
4) Example: "`branch`" is similar to "`account.branch`" as they both describe the bank branch associated with a relationship.
Formula:

```
similarity =
(number of similar items) / (total
items in descriptor_0)
```

Steps to Follow:
1) Compare each column (name and description) of `descriptor_24` with each column in `descriptor_0` to identify similar items.
2) Count the number of similar items.
3) Compute the similarity index using the given formula.
4) Return the computed value.

FIGURE 4. Prompt to compare `descriptor_24` against `descriptor_0`.

This asymmetry is intentional and driven by the prompt definition, where the denominator is always taken from the second descriptor.

Once all 600 comparisons were completed using GPT-4o, the resulting 25×25 similarity matrix (the elements in the diagonal have all value 1 as one descriptor has the highest similarity with itself) forms the gold standard. It is shown in Table 1.

The matrix is visualized using a color gradient: cells with the highest similarity scores appear in green, those with the lowest in red, and intermediate values are represented by varying shades between these extremes. This gradient enables an intuitive interpretation of semantic closeness between descriptor pairs based on their similarity scores, which range from 0 to 1.

VI. EXPERIMENTAL EVALUATION

The similarity matrix in Table 2 was generated using our embedding-based approach, with the same color gradient as the gold standard matrix.

TABLE 1. Gold standard 25 × 25 similarity matrix.

	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20	D21	D22	D23	D24
D0	1.00	1.00	1.00	0.94	1.00	0.14	0.14	0.00	0.25	0.00	0.11	0.31	0.12	0.22	0.10	0.17	0.10	0.40	0.50	0.50	0.50	0.50	0.30	0.50	0.67
D1	0.57	1.00	0.68	0.81	1.00	0.00	0.00	0.25	0.00	0.00	0.11	0.23	0.18	0.33	0.00	0.33	0.40	0.50	0.50	0.50	0.50	0.40	0.40	0.50	0.50
D2	0.73	0.88	1.00	0.94	0.82	0.29	0.00	0.00	0.00	0.00	0.00	0.23	0.24	0.44	0.20	0.7	0.2	0.4	0.6	0.4	0.4	0.5	0.3	0.6	0.5
D3	0.53	0.71	0.73	1.00	0.76	0.00	0.29	0.13	0.00	0.00	0.00	0.31	0.24	0.22	0.30	0.3	0.4	0.4	0.4	0.4	0.4	0.5	0.6	0.5	0.7
D4	0.50	1.00	0.68	0.88	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.15	0.18	0.22	0.10	0.7	0.4	0.7	0.7	0.7	0.6	0.3	0.4	0.5	0.7
D5	0.03	0.00	0.09	0.06	0.06	1.00	1.00	1.00	0.25	0.14	0.11	0.00	0.06	0.00	0.00	0.3	0.3	0.3	0.3	0.3	0.3	0.2	0.3	0.4	0.5
D6	0.07	0.06	0.14	0.00	0.06	1.00	1.00	0.88	0.25	0.14	0.11	0.15	0.18	0.22	0.20	0.1	0.4	0.3	0.1	0.1	0.2	0.1	0.3	0.4	0.3
D7	0.10	0.00	0.00	0.13	0.00	1.00	1.00	1.00	0.25	0.14	0.11	0.23	0.18	0.22	0.10	0.1	0.3	0.1	0.1	0	0.1	0	0.1	0.4	0.2
D8	0.03	0.00	0.05	0.00	0.06	0.14	0.14	0.13	1.00	0.57	0.22	0.00	0.06	0.11	0.00	0.1	0.1	0.2	0.2	0.2	0.2	0.1	0.1	0	0.2
D9	0.00	0.00	0.05	0.00	0.00	0.14	0.14	0.13	1.00	1.00	0.56	0.00	0.06	0.00	0.20	0.2	0.2	0.3	0.3	0.2	0.2	0.1	0.2	0	0.2
D10	0.03	0.00	0.00	0.00	0.00	0.14	0.14	0.13	0.50	0.71	1.00	0.15	0.06	0.11	0.10	0.1	0.2	0.3	0.2	0.2	0.2	0.1	0.1	0	0.2
D11	0.10	0.12	0.18	0.25	0.24	0.29	0.00	0.13	0.25	0.00	0.11	1.00	0.35	0.67	1.00	0.8	0	0.4	0.3	0.4	0.4	0.4	0.3	0.1	0.5
D12	0.20	0.24	0.14	0.31	0.35	0.14	0.14	0.13	0.25	0.29	0.11	0.50	1.00	1.00	0.70	0.50	0.10	0.10	0.20	0.80	0.80	0.70	0.20	0.25	0.50
D13	0.10	0.24	0.05	0.25	0.06	0.00	0.14	0.25	0.00	0.14	0.00	0.38	0.53	1.00	0.50	0.5	0	0.3	0.2	0.8	0.7	0.6	0.2	0.1	0.7
D14	0.07	0.00	0.09	0.19	0.29	0.00	0.14	0.00	0.25	0.29	0.00	0.77	0.24	0.44	1.00	0.8	0	0.1	0.2	0.5	0.7	0.5	0.4	0.25	0.7
D15	0.58	0.24	0.25	0.31	0.24	0.36	0.36	0.36	0.25	0.14	0.11	0.85	0.38	0.61	1.00	1.00	0.1	0.2	0.1	0.4	0.3	0.5	0.3	0.3	0.3
D16	0.2	0.3	0.3	0.3	0.2	0.3	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1.00	0.1	0.1	0.1	0.1	0.1	0.1	0.13	0.3
D17	0.4	0.4	0.4	0.5	0.5	0.1	0.1	0.3	0.3	0.3	0.1	0	0.4	0.4	0.5	0.4	0.1	1.00	0.1	0.4	0.4	0.5	0.1	0.5	0.2
D18	0.3	0.3	0.4	0.4	0.4	0.3	0.3	0.3	0.3	0.1	0.1	0.2	0.3	0.2	0.4	0.3	0.2	0.7	1.00	0.3	0.3	0.3	0.2	0.38	0.3
D19	0.17	0.24	0.27	0.31	0.24	0.29	0.29	0.25	0.25	0.14	0.11	0.46	0.47	0.56	0.50	0.42	0.10	0.40	0.30	1.00	0.60	0.60	0.10	0.25	0.17
D20	0.13	0.18	0.14	0.19	0.18	0.14	0.14	0.13	0.25	0.14	0.11	0.39	0.47	0.56	0.60	0.50	0.10	0.40	0.50	0.80	1.00	0.70	0.20	0.50	0.33
D21	0.10	0.06	0.14	0.19	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.31	0.41	0.56	0.40	0.42	0.00	0.00	0.10	0.70	0.70	1.00	0.10	0.25	0.33
D22	0.10	0.12	0.14	0.19	0.12	0.57	0.57	0.63	0.25	0.29	0.22	0.39	0.29	0.44	0.50	0.42	0.40	0.60	0.30	0.50	0.50	0.40	1.00	0.63	0.50
D23	0.33	0.06	0.05	0.06	0.06	0.14	0.14	0.13	0.00	0.00	0.00	0.08	0.12	0.22	0.20	0.17	0.00	0.20	0.40	0.30	0.40	0.30	0.40	1.00	0.50
D24	0.13	0.18	0.14	0.13	0.18	0.29	0.29	0.25	0.25	0.14	0.11	0.15	0.18	0.11	0.20	0.17	0.20	0.10	0.20	0.20	0.20	0.10	0.30	0.13	1.00

TABLE 2. Embedding-based model 25 × 25 similarity matrix.

	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20	D21	D22	D23	D24
D0	1	1	1	1	0.82	0.29	0.14	0.12	0.25	0.14	0	0.08	0.24	0.22	0.1	0.17	0.1	0	0	0	0	0.3	0	0.12	0.33
D1	0.57	1	0.64	0.75	0.88	0.14	0.14	0	0.25	0.14	0.11	0	0.24	0.22	0	0.17	0.1	0.1	0.2	0.2	0.2	0.2	0	0.12	0.33
D2	0.73	0.82	1	1	0.76	0.14	0	0	0.25	0.14	0	0	0.18	0.22	0.1	0.17	0.1	0	0.1	0.1	0.1	0.3	0	0.12	0.5
D3	0.53	0.71	0.73	1	0.76	0.14	0.14	0.12	0.25	0.14	0.11	0	0.24	0.33	0	0.17	0.1	0.1	0.2	0.1	0.2	0.3	0	0.12	0.17
D4	0.47	0.88	0.59	0.81	1	0.14	0.14	0.12	0.25	0.14	0.11	0	0.24	0.22	0	0.17	0.1	0.1	0.1	0.1	0.1	0.3	0	0.12	0.5
D5	0.07	0.06	0.05	0.06	0.06	1	1	0.88	0.25	0.14	0.22	0	0.06	0	0	0	0.1	0.1	0.1	0	0.1	0	0	0	0.17
D6	0.03	0.06	0	0.06	0.06	1	1	0.88	0.25	0.14	0.11	0	0.06	0	0	0	0.1	0.1	0.1	0	0.1	0	0	0.12	0.17
D7	0.03	0	0	0.06	0.06	1	1	1	0.25	0.14	0.11	0	0.06	0	0	0	0.1	0	0	0	0	0	0	0.12	0.17
D8	0.03	0.06	0.05	0.06	0.06	0.14	0.14	0.12	1	0.57	0.22	0	0.06	0	0	0	0.1	0.1	0.1	0	0.1	0	0	0.12	0.17
D9	0.03	0.06	0.05	0.06	0.06	0.14	0.14	0.12	1	1	0.56	0	0.06	0	0	0	0.1	0.1	0.1	0	0.1	0	0	0.12	0.17
D10	0	0.06	0	0.06	0.06	0.29	0.14	0.12	0.5	0.71	1	0	0.06	0	0	0	0.1	0.1	0	0	0.1	0	0.3	0.12	0.17
D11	0.03	0	0	0	0	0	0	0	0	0	0	1	0.24	0.33	1	0.83	0	0	0	0.2	0.2	0.1	0.3	0.25	0.17
D12	0.13	0.24	0.14	0.25	0.24	0.14	0.14	0.12	0.25	0.14	0.11	0.31	1	0.89	0.4	0.5	0.1	0.1	0.2	0.7	0.6	0.6	0.2	0.38	0.5
D13	0.07	0.12	0.09	0.19	0.12	0	0	0	0	0	0	0.23	0.47	1	0.3	0.42	0	0	0	0.4	0.2	0.4	0.2	0.38	0.17
D14	0.03	0	0.05	0	0	0	0	0	0	0	0	0.77	0.24	0.33	1	0.83	0	0	0	0.2	0.2	0.1	0.3	0.25	0.17
D15	0.07	0.12	0.09	0.12	0.12	0	0	0	0	0	0	0.77	0.35	0.56	1	1	0	0	0	0.3	0.2	0.2	0.3	0.38	0.17
D16	0.03	0.06	0.05	0.06	0.06	0.14	0.14	0.12	0.25	0.14	0.11	0	0.06	0	0	0	1	0.1	0.1	0	0.1	0	0	0.12	0.33
D17	0	0.06	0	0.06	0.06	0.14	0.14	0	0.25	0.14	0.11	0	0.06	0	0	0	0.1	1	0.1	0.1	0.1	0	0	0.38	0.17
D18	0	0.12	0.05	0.06	0.06	0.14	0.14	0	0.25	0.14	0	0	0.12	0	0	0	0.1	0.1	1	0.1	0.2	0	0	0.12	0.33
D19	0	0.12	0.05	0.06	0.06	0	0	0	0	0	0	0.15	0.41	0.44	0.2	0.25	0	0.1	0.1	1	0.6	0.5	0.2	0.38	0.17
D20	0	0.12	0.05	0.12	0.06	0.14	0.14	0	0.25	0.14	0.11	0.15	0.35	0.22	0.2	0.17	0.1	0.1	0.2	0.6	1	0.2	0.2	0.38	0.33
D21	0.1	0.12	0.14	0.19	0.18	0	0	0	0	0	0	0.08	0.35	0.44	0.1	0.17	0.1	0	0	0.5	0.2	1	0.1	0.12	0.17
D22	0	0	0	0	0	0	0	0	0	0	0	0.23	0.12	0.22	0.3	0.25	0.1	0	0	0.2	0.2	0.1	1	0.5	0.33
D23	0.03	0.06	0.05	0.06	0.06	0.14	0.14	0.12	0.25	0.14	0.11	0.15	0.18	0.33	0.2	0.25	0.1	0.3	0.1	0.3	0.3	0.1	0.4	1	0.33
D24	0.07	0.12	0.14	0.06	0.18	0.14	0.14	0.12	0.25	0.14	0.11	0.08	0.18	0.11	0.1	0.08	0.2	0.1	0.2	0.1	0.2	0.1	0.2	0.25	1

Finally, the matrix representing the absolute differences between the two matrices shown in Table 1 and Table 2 was computed as follows:

$$\texttt{DifferenceMatrix}[i][j] = \left|\texttt{GPTMatrix}[i][j] - \texttt{EmbeddingMatrix}[i][j]\right|$$

The difference matrix is shown in Table 3 and serves as an interpretability tool to assess the disagreement between our approach and the gold standard. Each cell represents the absolute difference between the corresponding similarity scores. A three-color scale is used for visualization:

- **Green** for high agreement / low disagreement (difference in $[0.00 - 0.25]$),
- **Yellow** for moderate agreement/disagreement (difference in $[0.26 - 0.50]$),
- **Red** for low agreement / high disagreement (difference > 0.50).

This color-coded representation allows for a quick visual inspection of how closely the output of our approach aligns with the gold standard reference.

As it can be noted, only 12 out of 600 values have a low agreement. To further analyze the results, Table 4 summarizes the performance metrics obtained by comparing the similarity matrix generated by our algorithm with the gold standard matrix, across multiple threshold values. Each threshold defines the minimum similarity score required for a descriptor pair to be considered *similar*.

Diagonal values (i.e., self-comparisons) are excluded from all computations, resulting in a total of $N = 600$ off-diagonal descriptor pairs in the 25×25 matrix. This exclusion prevents artificially inflated performance metrics, as all diagonal elements trivially score maximum similarity. In the following, we will describe each column of Table 4.

- **Threshold:** The similarity threshold value (t), ranging from 0.00 to 1. A descriptor pair is classified as *similar* if its similarity score is greater than or equal to t.
- **Percentage:** The proportion of off-diagonal cells in the algorithm's similarity matrix with scores equal to or above the underlying threshold t:

$$\text{Percentage} = \frac{\#\{\text{cells with similarity} > t\}}{600} \times 100$$

- **Mean Absolute Error (MAE):** The mean of the absolute differences between the similarity scores from the algorithm and the gold standard, considering only those elements in the algorithm's matrix that exceed the underlying threshold t:

$$\text{MAE}(t) = \frac{1}{N_t} \sum_{\substack{i \neq j \\ M_{\text{alg}}^{(i,j)} > t}} \left| M_{\text{LLM}}^{(i,j)} - M_{\text{alg}}^{(i,j)} \right|$$

where N_t is the number of elements above the threshold t.

TABLE 3. Gold Standard-embedding-based model 25 × 25 difference matrix.

	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20	D21	D22	D23	D24
D0	0.00	0.00	0.00	0.06	0.18	0.15	0.00	0.12	0.00	0.14	0.11	0.23	0.12	0.00	0.00	0.00	0.00	0.40	0.50	0.50	0.50	0.20	0.30	0.38	0.34
D1	0.00	0.00	0.04	0.06	0.12	0.14	0.14	0.25	0.25	0.14	0.00	0.23	0.06	0.11	0.00	0.16	0.30	0.40	0.30	0.30	0.30	0.20	0.40	0.38	0.17
D2	0.00	0.00	0.00	0.00	0.06	0.15	0.00	0.00	0.00	0.00	0.00	0.23	0.06	0.22	0.10	0.50	0.10	0.60	0.50	0.30	0.30	0.20	0.30	0.51	0.00
D3	0.00	0.00	0.00	0.00	0.00	0.14	0.15	0.01	0.25	0.14	0.11	0.31	0.01	0.11	0.30	0.16	0.30	0.30	0.20	0.30	0.20	0.20	0.60	0.38	0.50
D4	0.03	0.12	0.09	0.06	0.00	0.14	0.14	0.12	0.25	0.14	0.11	0.15	0.06	0.00	0.10	0.50	0.30	0.60	0.60	0.60	0.50	0.00	0.40	0.38	0.17
D5	0.04	0.06	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.00	0.25	0.20	0.20	0.20	0.30	0.20	0.20	0.30	0.38	0.33
D6	0.04	0.00	0.14	0.06	0.06	0.00	0.00	0.01	0.00	0.00	0.00	0.15	0.12	0.22	0.20	0.08	0.30	0.20	0.00	0.10	0.10	0.10	0.30	0.26	0.16
D7	0.07	0.00	0.00	0.07	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.23	0.12	0.22	0.10	0.08	0.20	0.10	0.10	0.00	0.10	0.00	0.10	0.12	0.00
D8	0.00	0.06	0.01	0.06	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.08	0.00	0.10	0.10	0.20	0.10	0.10	0.10	0.12	0.00
D9	0.03	0.06	0.00	0.06	0.06	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.17	0.10	0.20	0.20	0.20	0.10	0.10	0.20	0.12	0.00
D10	0.03	0.06	0.00	0.06	0.06	0.15	0.00	0.01	0.00	0.00	0.00	0.15	0.00	0.11	0.10	0.08	0.10	0.20	0.20	0.20	0.10	0.10	0.10	0.12	0.00
D11	0.07	0.12	0.18	0.25	0.24	0.29	0.00	0.13	0.25	0.00	0.11	0.00	0.11	0.34	0.00	0.00	0.00	0.40	0.30	0.20	0.20	0.30	0.00	0.12	0.33
D12	0.07	0.00	0.00	0.06	0.11	0.00	0.01	0.25	0.00	0.15	0.00	0.19	0.00	0.11	0.30	0.00	0.00	0.00	0.00	0.40	0.20	0.10	0.00	0.13	0.00
D13	0.03	0.12	0.04	0.06	0.06	0.14	0.14	0.25	0.00	0.14	0.00	0.15	0.06	0.00	0.20	0.08	0.00	0.30	0.20	0.40	0.50	0.20	0.00	0.25	0.50
D14	0.04	0.00	0.04	0.19	0.29	0.00	0.14	0.00	0.25	0.29	0.00	0.00	0.01	0.11	0.00	0.00	0.00	0.10	0.20	0.30	0.50	0.40	0.10	0.00	0.50
D15	0.51	0.12	0.16	0.19	0.12	0.36	0.36	0.36	0.25	0.14	0.11	0.08	0.03	0.05	0.00	0.00	0.10	0.20	0.10	0.10	0.10	0.30	0.00	0.13	0.16
D16	0.14	0.23	0.25	0.21	0.17	0.13	0.06	0.05	0.18	0.07	0.01	0.10	0.04	0.10	0.13	0.13	0.00	0.00	0.00	0.10	0.00	0.07	0.10	0.01	0.00
D17	0.41	0.37	0.35	0.44	0.47	0.00	0.00	0.29	0.00	0.15	0.01	0.00	0.29	0.42	0.45	0.42	0.00	0.00	0.00	0.34	0.30	0.50	0.10	0.12	0.00
D18	0.27	0.15	0.32	0.26	0.29	0.15	0.15	0.25	0.00	0.00	0.11	0.23	0.17	0.22	0.40	0.33	0.10	0.60	0.00	0.23	0.10	0.30	0.20	0.26	0.00
D19	0.17	0.12	0.22	0.25	0.18	0.29	0.29	0.25	0.25	0.14	0.11	0.31	0.06	0.12	0.30	0.17	0.10	0.30	0.20	0.00	0.00	0.10	0.10	0.13	0.00
D20	0.13	0.06	0.09	0.07	0.12	0.00	0.00	0.13	0.00	0.00	0.00	0.24	0.12	0.34	0.40	0.33	0.00	0.30	0.30	0.20	0.00	0.50	0.00	0.12	0.00
D21	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.23	0.06	0.12	0.30	0.25	0.00	0.00	0.10	0.20	0.50	0.00	0.00	0.13	0.16
D22	0.10	0.12	0.14	0.19	0.12	0.57	0.57	0.63	0.25	0.29	0.22	0.16	0.17	0.22	0.20	0.17	0.40	0.60	0.30	0.30	0.30	0.30	0.00	0.13	0.17
D23	0.30	0.00	0.00	0.00	0.00	0.14	0.00	0.01	0.25	0.14	0.11	0.07	0.06	0.11	0.00	0.08	0.10	0.10	0.30	0.00	0.10	0.20	0.00	0.00	0.17
D24	0.06	0.06	0.00	0.07	0.00	0.15	0.15	0.13	0.00	0.00	0.00	0.07	0.00	0.00	0.10	0.09	0.00	0.00	0.00	0.10	0.00	0.00	0.10	0.13	0.00

- **Mean Square Error (MSE):** The corresponding squared error metric:

$$\mathrm{MSE}(t) = \frac{1}{N_t} \sum_{\substack{i \neq j \\ M_{\mathrm{alg}}^{(i,j)} > t}} \left(M_{\mathrm{LLM}}^{(i,j)} - M_{\mathrm{alg}}^{(i,j)} \right)^2$$

- **Precision:** Measures the proportion of *true positives* among all predicted similar pairs for the underlying threshold t:

$$\mathrm{Precision}(t) = \frac{\#\{(i,j) : M_{\mathrm{alg}}^{(i,j)} > t \wedge M_{\mathrm{LLM}}^{(i,j)} > t\}}{\#\{(i,j) : M_{\mathrm{alg}}^{(i,j)} > t\}}$$

- **Recall:** Measures the proportion of correctly retrieved similar pairs with respect to the gold standard for the underlying threshold t:

$$\mathrm{Recall}(t) = \frac{\#\{(i,j) : M_{\mathrm{alg}}^{(i,j)} > t \wedge M_{\mathrm{LLM}}^{(i,j)} > t\}}{\#\{(i,j) : M_{\mathrm{LLM}}^{(i,j)} > t\}}$$

- **F1 Score:** The harmonic mean of Precision and Recall for the underlying threshold t:

$$\mathrm{F1}(t) = 2 \cdot \frac{\mathrm{Precision}(t) \cdot \mathrm{Recall}(t)}{\mathrm{Precision}(t) + \mathrm{Recall}(t)}$$

- **Accuracy:** Accuracy was computed as the proportion of correctly classified descriptor pairs, those for which both the algorithm and the gold standard agree on being either similar or dissimilar, over the total number of pairs for the underlying threshold t:

$$\mathrm{Accuracy}(t) = \frac{\mathrm{TP}(t) + \mathrm{TN}(t)}{\mathrm{TN}(t) + \mathrm{TP}(t) + \mathrm{FP}(t) + \mathrm{FN}(t)}$$

  True positives (TP) and true negatives (TN) are determined by whether the similarity scores from both matrices are above or below t, respectively.
- **# Similar:** The number of descriptor pairs (excluding diagonals) with similarity equal to or greater than the threshold in the algorithm's matrix.
- **# Different:** The number of descriptor pairs with similarity scores less than the threshold. The sum of these two last fields is always 600.

This evaluation enables a nuanced understanding of the algorithm's behavior at varying levels of similarity strictness, showing how precision improves as the threshold increases, while recall typically decreases.

As expected, when the threshold is set to 0, all elements from our approach are retrieved, since all similarity scores are by definition greater than or equal to 0. This results in full recall but no discrimination.

As we begin to increase the threshold, the number of retrieved elements decreases. Initially, this leads to a drop in F1 score, which then increases as precision improves, before dropping again at higher thresholds. Accuracy follows a similar trend: it decreases at first, but starts rising earlier than the F1 score and continues to increase up to the highest threshold values.

Excluding the trivial case with threshold 0, the optimal threshold in terms of both F1 score and accuracy is 0.75. At this threshold, 26 elements from the algorithm's similarity matrix have values equal to or above 0.75. Since the precision at this point is 1, it means that all 26 of these elements also have values equal to or above 0.75 in the gold standard matrix; no false positives were introduced.

However, the recall at this threshold is 0.87, which indicates that 4 relevant elements are missing, specifically, there are 4 elements in the gold standard matrix with similarity values equal to or above 0.75 that were not matched by the algorithm (i.e., their corresponding values in the algorithm's matrix were below the threshold).

VII. ERROR ANALYSIS

To better understand the limitations of the model, we present three representative examples of different types of errors observed during evaluation. Each case highlights a specific discrepancy between the similarity values computed by our approach and those provided by the LLM-based gold standard:

- an example where our approach fails to detect a moderately strong similarity identified by the gold standard;
- a case where our approach underestimates the number of matching items;
- and one where it overestimates similarity due to superficial matches.

A. FIRST EXAMPLE: UNDETECTED SEMANTIC SIMILARITY

An illustrative example of a significant discrepancy is found in the pair `descriptor_22` and `descriptor_7`, which exhibits the highest absolute difference value in the difference matrix (0.63). According to the gold standard matrix, this pair received a similarity score of 0.63, indicating a moderate conceptual similarity. In contrast, our approach assigned a similarity score of 0.00, completely failing to capture this relationship.

To better understand this divergence, we inspected the detailed alignment log produced by the LLM for this pair. The log highlights multiple weak or partial correspondences between attributes such as `investor_id` and `client_identifier`, `asset_type` and `type_client`, and `market_value` and `est_revenue`. While these columns are not exact matches, they reflect semantically related concepts that the LLM is able to interpret and link based on contextual knowledge and real-world associations.

However, our approach relies on surface-level signals or rigid structural similarities (e.g., token overlap or fixed rules), and therefore, fails to recognize these nuanced semantic relationships. This case exemplifies one of the key limitations of the algorithm: its inability to generalize or infer similarity when there is lexical or structural variation that does not follow predefined patterns.

TABLE 4. Threshold-based evaluation metrics comparing the algorithm's similarity matrix to the gold standard.

Threshold	Percentage	Mean Absolute Error	Mean Square Error	Precision	Recall	F1	Accuracy	#Similar	#Different
0.00	100.00	0.14	0.04	1.00	1.00	1.00	1.00	600	0
0.05	69.33	0.14	0.04	0.91	0.72	0.80	0.69	416	184
0.10	48.67	0.13	0.04	0.90	0.53	0.67	0.57	292	308
0.15	33.00	0.11	0.07	0.91	0.49	0.63	0.65	198	402
0.20	22.67	0.10	0.04	0.88	0.35	0.50	0.60	136	464
0.25	19.67	0.08	0.03	0.87	0.37	0.52	0.68	118	482
0.30	15.67	0.07	0.03	0.93	0.38	0.54	0.75	94	506
0.35	11.50	0.04	0.02	0.94	0.35	0.51	0.80	69	531
0.40	10.00	0.04	0.02	1.00	0.36	0.53	0.82	60	540
0.45	8.67	0.03	0.01	1.00	0.42	0.59	0.88	52	548
0.50	8.33	0.02	0.01	1.00	0.42	0.59	0.88	50	550
0.55	6.83	0.01	0.01	1.00	0.53	0.69	0.94	41	559
0.60	6.00	0.01	0.00	1.00	0.53	0.69	0.95	36	564
0.65	5.17	0.00	0.00	1.00	0.56	0.72	0.96	31	569
0.70	5.17	0.00	0.00	1.00	0.69	0.82	0.98	31	569
0.75	4.33	0.00	0.00	1.00	0.87	0.93	0.99	26	574
0.80	3.50	0.00	0.00	1.00	0.75	0.86	0.99	21	579
0.85	2.67	0.00	0.00	1.00	0.80	0.89	0.99	16	584
0.90	1.83	0.00	0.00	1.00	0.69	0.81	0.99	11	589
0.95	1.83	0.00	0.00	0.82	0.64	0.72	0.99	11	589
1.00	1.83	0.00	0.00	0.82	0.64	0.72	0.99	11	589

Such findings confirm the importance of incorporating semantic understanding, potentially through fine-tuned language models or ontology-based mapping, in order to enhance alignment quality and reduce such interpretability gaps.

B. SECOND EXAMPLE: UNDERESTIMATION OF SIMILAR FIELDS

In the comparison between `descriptor_4` and `descriptor_15`, our approach significantly underestimated the similarity score (0.17 for our approach in Table 2 vs. 0.67 for the gold standard in Table 1). It failed to identify multiple semantically similar fields, such as `operation_type` vs. `operationType`, `last_update` vs. `applicationTimestamp`, and `deal_identifier` vs. `transactionId`. These pairs were marked as dissimilar (dissimilarity > 0.3), although they clearly refer to the same operational concepts.

The discrepancy likely stems from our approach's reliance on lexical features in `name` and `description`, which are often insufficient to capture synonymy or contextual meaning. In contrast, the LLM-based gold standard recognized these semantic relationships through contextual reasoning and domain knowledge.

C. THIRD EXAMPLE: OVERESTIMATION DUE TO SUPERFICIAL MATCHES

In the comparison between `descriptor_13` and `descriptor_23`, our approach overestimated similarity (0.38 for our approach in Table 2 vs. 0.1 for the gold standard in Table 1), incorrectly matching fields such as `Amount`, which refers to a *monetary value* within a financial context (e.g., the amount of money involved in a transaction, as suggested by the descriptor's overall theme and description), with `quantity`, which instead denotes a *unit count*. Similarly, it matched `operation_id` with `user`, failing to capture their distinct semantic roles.

Again, these mismatches highlight the model's limited capacity to integrate broader contextual cues when comparing fields in isolation. These overestimations are likely due to shallow lexical overlaps or structural similarities (e.g., the presence of "id"), despite the underlying semantic mismatch. The gold standard appropriately rejected these matches due to their different functional roles in the dataset.

These cases underscore the limitations of using surface-level lexical similarity in evaluating schema alignment tasks and highlight the value of integrating deeper semantic models or hybrid approaches that combine statistical and symbolic reasoning.

VIII. COMPUTATIONAL ANALYSIS

While the previous section highlighted some of the key limitations of our approach, particularly its inability to capture nuanced semantic relationships identified by the LLM-based gold standard, it is important to underscore the primary design objective of our system: efficiency. Our method was intentionally developed to prioritize speed and lightweight computation over deep semantic understanding. This trade-off is deliberate and grounded in practical constraints: the tool is intended to run locally, without relying on external APIs or large-scale language models, and must deliver near-instant results to support rapid, iterative development workflows. In the following, we evaluate the computational performance of our system and demonstrate that, despite some compromises in accuracy, it achieves substantial gains in speed and scalability, making it well-suited for environments where responsiveness and low resource consumption are critical.

As we know, the LLM-based solution offers remarkable accuracy and contextual understanding when computing similarity values between descriptors, mainly due to its extensive training data and powerful contextual reasoning capabilities. As a result, this model is highly effective at capturing nuanced semantic similarities, providing reliable and robust similarity measurements.

However, the primary limitation of the LLM-based solution lies in its computational efficiency. Specifically, calculating each similarity request with ChatGPT's API requires approximately 6 seconds. Thus, computing all 600 similarity values using this approach would theoretically take around 3600 seconds (approximately 60 minutes). This considerable computational overhead makes the LLM-based approach significantly less practical for scenarios demanding real-time or near-real-time processing.

Conversely, our embedding-based model demonstrates a substantial computational advantage. Our experiments indicate that calculating all 600 similarity values requires less than 23 seconds, making our solution 156 times faster than the LLM-based alternative. Moreover, the computational resources needed by our model are modest, enabling its efficient execution even on standard personal computers without specialized hardware or cloud-based resources.

In conclusion, although the LLM-based model provides exceptional accuracy in semantic similarity due to its deep contextual understanding, its practical applicability is hindered by prohibitively high computational demands. Our embedding-based model, by contrast, offers an optimal trade-off between computational efficiency and accuracy, making it significantly more suitable for applications where timely results and resource constraints are critical factors.

IX. CONCLUSION

In this work, we presented the Data Product Similarity system, a lightweight and efficient solution designed to identify potential duplication among Data Products within an organization. Unlike LLM-based approaches that provide deep semantic understanding but come with substantial computational overhead, our system is intentionally optimized for speed, simplicity, and operational feasibility. By relying on schema-level structural features and employing the Hungarian algorithm to determine optimal field alignment, the system enables fast and practical similarity assessments that can be integrated directly into local development environments.

Through experimental evaluation against a gold standard generated via a state-of-the-art LLM, we demonstrated that our approach is capable of achieving competitive performance in many cases while significantly reducing processing time. Our error analysis highlighted typical failure modes, especially in scenarios requiring nuanced semantic interpretation, a known limitation of purely structural approaches. However, these trade-offs are acceptable and even desirable in contexts where responsiveness and low resource consumption are critical, such as during early-stage Data Product development or in organizations with constrained infrastructure.

Future work may explore hybrid solutions that retain the efficiency of our current system while integrating lightweight semantic enrichment techniques, such as domain-specific ontologies or compact embedding models. Ultimately, our results suggest that even in the absence of full semantic modeling, substantial value can be delivered through fast, interpretable, and scalable tools that promote reuse and reduce duplication in modern data ecosystems.

REFERENCES

[1] I. Blohm, F. Wortmann, C. Legner, and F. Köbler, "Data products, data mesh, and data fabric," *Bus. Inf. Syst. Eng.*, vol. 66, no. 5, pp. 643–652, Oct. 2024, doi: 10.1007/s12599-024-00876-5.

[2] Z. Dehghani, *Data Mesh: Delivering Data Driven Value At Scale*. O'Reilly Media, 2022.

[3] A. Fawzy, A. Tahir, M. Galster, and P. Liang, "Exploring data management challenges and solutions in agile software development: A literature review and practitioner survey," *Empirical Softw. Eng.*, vol. 30, no. 3, p. 77, May 2025, doi: 10.1007/s10664-025-10630-4.

[4] J. Bode, N. Kühl, D. Kreuzberger, and C. Holtmann, "Toward avoiding the data mess: Industry insights from data mesh implementations," *IEEE Access*, vol. 12, pp. 95402–95416, 2024, doi: 10.1109/ACCESS.2024.3417291.

[5] R. Shah, K. Mukherjee, A. Tyagi, S. K. Karnam, D. Joshi, S. P. Bhosale, and S. Mitra, "R2D2: Reducing redundancy and duplication in data lakes," *Proc. ACM Manage. Data*, vol. 1, no. 4, pp. 1–25, Dec. 2023, doi: 10.1145/3626762.

[6] A. Qidwai, S. Mukhopadhyay, P. Khatiwada, D. Roth, and V. Gupta, "PRAISE: Enhancing product descriptions with LLM-driven structured insights," in *Proc. 63rd Annu. Meeting Assoc. Comput. Linguistics*, Jul. 2025, pp. 644–652. [Online]. Available: https://aclanthology.org/2025.acl-demo.62/

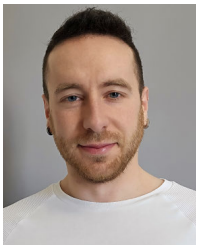
[7] H. W. Kuhn, "The Hungarian method for the assignment problem," *Nav. Res. Logistics (NRL)*, vol. 52, no. 1, pp. 7–21, Feb. 2005.

[8] M. Yu, G. Li, D. Deng, and J. Feng, "String similarity search and join: A survey," *Frontiers Comput. Sci.*, vol. 10, no. 3, pp. 399–417, Jun. 2016.

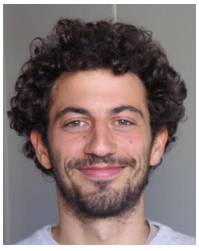
[9] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller, "JOSIE: Overlap set similarity search for finding joinable tables in data lakes," in *Proc. Int. Conf. Manage. Data*, Jun. 2019, pp. 847–864.

[10] Y. Dong, K. Takeoka, C. Xiao, and M. Oyamada, "Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, Apr. 2021, pp. 456–467.

[11] Y. Dong, C. Xiao, T. Nozawa, M. Enomoto, and M. Oyamada, "DeepJoin: Joinable table discovery with pre-trained language models," *Proc. VLDB Endowment*, vol. 16, no. 10, pp. 2458–2470, Jun. 2023. [Online]. Available: https://www.vldb.org/pvldb/vol16/p2458-dong.pdf
[12] J. Flores, S. Nadal, and O. Romero, "Effective and scalable data discovery with NextiaJD," in *Proc. Adv. Database Technology, EDBT*, Mar. 2021, pp. 690–693.
[13] S. Bharadwaj, P. Gupta, R. Bhagwan, and S. Guha, "Discovering related data at scale," *Proc. VLDB Endowment*, vol. 14, no. 8, pp. 1392–1400, Apr. 2021.
[14] A. Bogatu, A. A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, Apr. 2020, pp. 709–720.
[15] P. Ouellette, A. Sciortino, F. Nargesian, B. G. Bashardoost, E. Zhu, K. Q. Pu, and R. J. Miller, "RONIN: Data lake exploration," *Proc. VLDB Endowment*, vol. 14, no. 12, pp. 2863–2866, Jul. 2021.
[16] J. Freire, G. Fan, B. Feuer, C. Koutras, Y. Liu, E. H. M. Peña, A. Santos, C. T. Silva, and E. Wu, "Large language models for data discovery and integration: Challenges and opportunities," *IEEE Data Eng. Bull.*, vol. 48, no. 1, pp. 3–18, Jan. 2025.
[17] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," *Proc. VLDB Endowment*, vol. 11, no. 7, pp. 813–825, Mar. 2018.
[18] A. Khatiwada, G. Fan, R. Shraga, Z. Chen, W. Gatterbauer, R. J. Miller, and M. Riedewald, "SANTOS: Relationship-based semantic table union search," *Proc. ACM Manage. Data*, vol. 1, no. 1, pp. 1–25, May 2023.
[19] X. Hu, S. Wang, X. Qin, C. Lei, Z. Shen, C. Faloutsos, A. Katsifodimos, G. Karypis, L. Wen, and P. S. Yu, "Automatic table union search with tabular representation learning," in *Proc. Findings Assoc. Comput. Linguistics, ACL*, 2023, pp. 3786–3800.
[20] M. Ailem, J. Liu, and R. Qader, "Encouraging neural machine translation to satisfy terminology constraints," 2021, *arXiv:2106.03730*.
[21] D. König, "Graphs and matrices," *Matematikaiés Fizikai Lapok*, vol. 38, pp. 116–119, Jan. 1931.
[22] J. Egerváry, "Matrixok kombinatorikus tulajdonságairó," *Matematikaies Fizikai Lapok*, vol. 38, pp. 116–119, Feb. 1931.
[23] (2024). *Nltk Tokenizer Api Documentation*. [Online]. Available: https://www.nltk.org/api/nltk.tokenize.html
[24] N. Reimers and I. Gurevych. (2021). *All-Minilm-l6-V2*. [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2



ANTONIO D'AMBROSIO received the bachelor's degree (Hons.) in computer science and technologies for software production from the University of Bariis and the master's degree (Hons.) in artificial intelligence and information systems from the University of Turin. He is a Senior Software Engineer and a Project Lead of Witboost with Agile Lab, where he leads the design, development, and delivery of scalable solutions for data engineering and Data Mesh architectures. Within Witboost, a modular platform for Data Product Management, he was a Tech Lead, a Team Leader, and a Delivery Manager, coordinating cross-functional teams to ensure robust implementations of enterprise-level data strategies. His background bridges software engineering, big data, and AI-driven architectures, positioning him at the intersection of academic research and industry innovation.



PAOLO PLATTER received the M.S. degree (Hons.) in telecommunications engineering from Politecnico di Torino. He is the Co-Founder and the Chief Technology Officer of Agile Lab (Turin, Italy), where he leads architecture, operations, and innovation in scalable Big Data and AI solutions. He is also a Product Manager of Witboost, a platform designed for Data Product Management. His leadership has driven multiple enterprise-level Data Mesh implementations and strategic Data Engineering initiatives across Europe. His research contributions include ontology-based generation of data platform assets, underscoring his expertise in semantic-driven data architectures.

MARTINA SALIS received the bachelor's and master's degrees in computer science from the University of Cagliari, Italy, in 2019 and 2023, respectively. She is currently a Research Fellow with the University of Cagliari, working on the DAMPAI Project. Her research interests include artificial intelligence and data modeling, with a particular focus on the development of machine learning methods for data analysis and knowledge extraction in complex domains.

FRANCESCO SIMBOLA received the degree in computer science from the University of Cagliari, Italy, in 2024, where he is currently pursuing the master's degree in computer science. Since June 2024, he has been a Contractor of the DAMPAI Project.

DIEGO REFORGIATO RECUPERO received the Ph.D. degree in computer science from the University of Naples Federico II, Italy, in 2004. He has been a Full Professor with the Department of Mathematics and Computer Science, the University of Cagliari, Italy, since February 2022. From 2005 to 2008, he was a Postdoctoral Researcher with the University of Maryland College Park, USA. He co-founded seven companies within the ICT sector and is actively involved in European projects and research (with one of his companies he won more than 40 FP7 and H2020 projects). He is the author of more than 270 conference and journal papers in these research fields, with more than 4800 citations. His current research interests include sentiment analysis, semantic web, natural language processing, human-robot interaction, financial technology, and smart grid. He won different awards in his career (such as the Marie Curie International Reintegration Grant, the Marie Curie Innovative Training Network, the Best Researcher Award from the University of Catania, the Computer World Horizon Award, the Telecom Working Capital, the Startup Weekend, the Best Paper Award).

DANIELE RIBONI received the Ph.D. degree in computer science from the University of Milano, in 2007. He was a Postdoctoral Fellow and an Assistant Professor with the University of Milano. He has been an Associate Professor of computer science with the University of Cagliari, since 2015. His research interests includes activity recognition, pervasive healthcare, knowledge management, and privacy issues in pervasive, and mobile computing. He served as the TPC chair and the TPC vice-chair for different conferences and workshops in the field, including IEEE PerCom and the International Conference on Intelligent Environments. His contributions appear in major conferences and journals.