Inferring Missing Data Lineage Links from Schema Metadata Using Transformer-Based Models (Regular Paper)

Maciej Brzeski
Faculty of Mathematics and Computer Science
Jagiellonian University
Informatica
Kraków, Poland
maciej.brzeski@doctoral.uj.edu.pl

Adam Roman
Faculty of Mathematics and Computer Science
Jagiellonian University
Kraków, Poland
adam.roman@uj.edu.pl

ABSTRACT

Data lineage inference is essential for understanding and managing data flow within complex information systems. However, existing lineage extraction methods—often based on instrumentation, runtime analysis, or annotated pipelines—fail to address cases where such data is partially missing or unavailable. In this work, we introduce a scalable two-step approach for inferring missing data lineage links using only schema metadata. Inspired by techniques from schema matching, our method combines a bi-encoder for efficient candidate filtering with a cross-encoder for high-precision prediction. Unlike classical schema matching, which focuses on semantic equivalence, we model directional lineage relationships, identifying where one column is likely derived from another. This makes our method particularly suitable for environments where full lineage instrumentation is impractical—such as hybrid clouds, legacy systems, or post hoc audits. We evaluate our model on large-scale commercial datasets with millions of schema elements and extreme class imbalance—where only one in 90,000 candidate pairs is a valid link. In this setting, reducing false positives is crucial, and our model demonstrates high effectiveness while remaining computationally efficient.

VLDB Workshop Reference Format:
Maciej Brzeski and Adam Roman. Inferring Missing Data Lineage Links from Schema Metadata Using Transformer-Based Models (Regular Paper). VLDB 2025 Workshop: Applied AI for Database Systems and Applications (AIDB 2025).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/maciejbrzeski-uj/data-lineage-dataset.

1 INTRODUCTION

Data lineage is an essential concept in managing data-driven processes, being part of the data governance aspect of general data management. It provides a structured description of the process by which data is traversed from its acquisition to its various applications, offering insights into its provenance, transformations, and dependencies along the way [11, 17]. In an era of relentless data generation and utilization across diverse domains, understanding data lineage has emerged as a crucial endeavor with far-reaching implications.

At its core, data lineage embodies the principle of transparency, showing the pathways through which data flows within complex ecosystems of modern information architectures. By delineating the origins of data, capturing its transformations through computational and analytical processes, and tracing its lineage across systems, data lineage facilitates critical tasks such as data quality assessment, regulatory compliance, impact analysis, and identifying the source of information, enabling trust in data [38].

The significance of data lineage extends across industries and sectors, finding particular resonance in domains where data integrity, reliability, and reproducibility are paramount [12, 13, 15, 34]. In fields such as finance, healthcare, and scientific research, where decisions depend on the accuracy of data-driven insights, robust data lineage frameworks play a crucial role in establishing trust, mitigating risks, and enhancing accountability.

However, the pursuit of comprehensive data lineage is not without its challenges. The sheer volume, velocity, and variety of data generated in contemporary environments pose formidable obstacles to effectively capturing and managing lineage information. Moreover, the distributed nature of data ecosystems, coupled with evolving regulatory landscapes and technological complexities, further complicate efforts to trace lineage across heterogeneous environments.

In this article, we address the challenge of inferring missing data lineage links in large-scale systems where complete lineage information is unavailable or infeasible to obtain. Our approach introduces a scalable natural language processing method based on a two-step transformer architecture. Unlike previous methods that rely on execution-level instrumentation or end-to-end DAG analysis, our model is designed to infer lineage relationships based solely on schema-level metadata, such as fully qualified column paths (e.g., `schema.table.column`).

This design enables our method to operate in fragmented or legacy environments where traditional lineage solutions break down. It requires no runtime integration, and yet provides high precision in identifying directional lineage links. We compare our method with existing approaches and demonstrate its scalability and effectiveness on real-world commercial datasets containing millions of schema elements.

The structure of this paper is as follows. In Section 2 we introduce the problem of data lineage reconstruction and motivate the need for inference methods. Section 3 formulates the task of lineage

Proceedings of the VLDB Endowment. ISSN 2150-8097.

inference from schema metadata and outlines the key assumptions and challenges. Section 4 reviews related work in schema matching, filtering, and representation learning. Section 5 discusses how our approach differs from prior work and introduces the constraints and trade-offs that guide our model design. Section 6 covers methods for learning semantic similarity, from classical embeddings to transformer-based architectures. Section 7 presents our complete two-stage inference model. Section 8 reports on the experimental evaluation using real-world industrial datasets. Finally, Section 9 concludes the paper and outlines directions for future work.

2 RECONSTRUCTING DATA LINEAGE

2.1 Motivation and Practical Challenges

Reconstructing data lineage—that is, recovering the provenance, transformations, and dependencies of data assets across systems—is a central challenge in modern data governance. According to the 2025 Gartner Magic Quadrant for Data and Analytics Governance Platforms [43], data lineage must provide broad and deep support for tracking how data moves and evolves. However, Gartner also notes that lineage implementations are often incomplete or fragmented, due to technical complexity, scale, and cross-platform heterogeneity.

This difficulty is well known to practitioners. Enterprise clients using commercial lineage platforms frequently report missing or broken lineage segments, especially across system boundaries or manual processes. These gaps are often caused by limitations in runtime instrumentation, proprietary tools, or fragmented pipelines. In consequence, organizations face tangible risks in trust, compliance, and operational traceability—particularly when lineage is used to support audits or automated impact analysis.

These practical constraints motivate the search for lineage inference methods: techniques that aim to reconstruct missing lineage using metadata alone. The rest of this section reviews the technical barriers to reconstructing lineage from code and logs, and motivates our metadata-based approach.

Figure 1 illustrates a simplified but representative lineage graph in a modern enterprise environment. The final `SALES_PM_DEMO` dataset is derived from intermediate tables such as `SALES_REP` and `WEB_SALES`, which in turn are produced through a combination of ETL processes and reporting tools. These transformations span multiple platforms—Oracle Database, Databricks, Power BI—and often rely on domain-specific business logic, runtime parameters, or external scripts. While the logical dependency path may be clear to the engineers who created the flow, capturing it automatically across these systems remains a major challenge.

Modern data lineage systems typically rely on automatic extraction techniques to capture dependencies and transformations. These include parsing SQL and notebook code, analyzing workflow definitions from orchestrators (e.g., Airflow, dbt), and collecting execution logs from data platforms. This scanner-based approach has proven effective for well-structured, orchestrated pipelines where all transformations are declarative and instrumentation is available.

Several metadata-driven lineage tools have also been proposed to address these challenges, both in academia and industry [35]. However, practical deployments show that full lineage extraction requires more than metadata—it often depends on runtime context, configuration values, and execution traces. These are difficult to obtain, especially in distributed or legacy environments. Studies focusing on specific systems—such as Apache Spark [41] or ORM-based Python frameworks [18]—demonstrate the feasibility of fine-grained lineage tracking, but lack generality.

A large portion of logic in production pipelines is embedded in general-purpose programming languages like Python, Java, or C++. These segments are often opaque to scanners—either because they are dynamically executed, externally managed, or not parsable by current tools. As a result, lineage graphs are frequently incomplete, even when state-of-the-art tooling is in place.

Most traditional approaches to lineage reconstruction fall into three technical categories: static code analysis, metadata annotations, or runtime log parsing [4, 9, 44]. Each of these faces practical limitations. Static analysis struggles with dynamically typed or generated code. Annotation-based systems require manual effort and are rarely comprehensive. Runtime tracing, while powerful, depends on full instrumentation and consistent observability across platforms.

As a result, many lineage graphs remain incomplete—even when advanced tooling is available. In these cases, the problem is not theoretical ambiguity, but practical inaccessibility: the lineage exists in principle, but cannot be recovered through traditional extraction methods.

2.2 Data Lineage Inference

This motivates a different approach: rather than extract lineage from code or logs, we infer it from available metadata. By analyzing schema-level descriptors—such as table and column paths, naming conventions, and relational structure—we aim to identify likely lineage links using statistical modeling.

Only a few prior efforts have explored lineage inference in this sense. In TRACER [16], the authors infer primary and foreign keys using random forests, based on features like inclusion dependencies and name similarity. They then use these inferred keys to suggest lineage relationships. However, their method requires access to the data itself and assumes a narrow transformation space, limiting its applicability.

Another example is RELIC [39], which detects joins, groupings, and pivots using sampled data to reconstruct dependency trees. This approach also relies on having access to instance-level data and becomes inefficient at scale. Moreover, it assumes consistent naming conventions and deterministic behavior—conditions rarely satisfied in heterogeneous, real-world pipelines.

Our work takes a different path. We propose a method that infers lineage links purely from schema metadata, without relying on access to runtime, code, or data. The next section defines this task in more detail and explains how we model it using only schema metadata.

This need for inference arises not only from technical limitations, but also from the operational complexity of large organizations. Legacy systems, undocumented data flows, and fragmented ownership often prevent full lineage extraction, even when instrumentation is available. In such environments, metadata-based inference provides a lightweight and broadly applicable complement to existing tools.

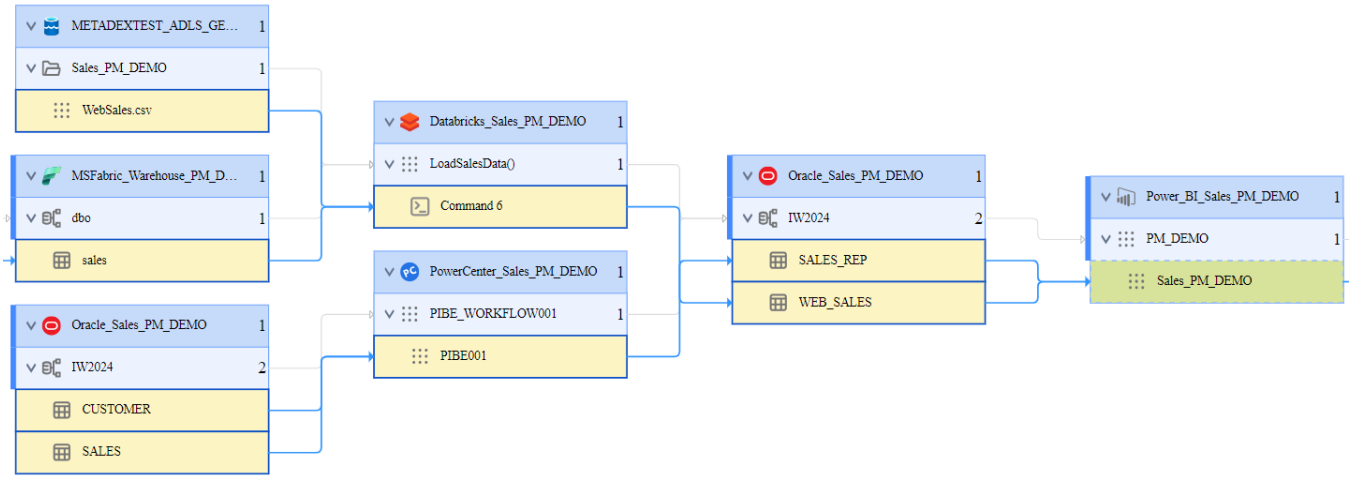


Figure 1: Example of data lineage

3 MODELING LINEAGE INFERENCE FROM SCHEMA METADATA

In this section, we describe the formulation and motivation behind our metadata-based approach to data lineage inference. Our goal is to infer lineage links using only schema-level metadata, rather than code, data values, or runtime traces. This reflects real-world constraints: in many enterprise environments, direct access to data is restricted, and lineage must be inferred across loosely connected subsystems that follow different domain models.

Enterprise data warehouses typically integrate information from heterogeneous sources originating from different departments and spanning domains such as sales, logistics, or production. These areas often operate as distinct organizational domains, each with its own semantics, naming conventions, and tools. As a result, global or end-to-end lineage modeling is not only technically challenging, but also conceptually inconsistent. The lineage of data across such boundaries may follow different logics and transformation paradigms, making unified modeling infeasible or misleading.

Instead of aiming to reconstruct complete cross-domain lineage, we focus on capturing localized, process-specific transformations using accessible metadata. Instance-level data access may be limited due to governance or operational constraints, and even when available, tables often contain millions of records. To ensure portability and minimal assumptions, we begin with a robust baseline: structured column paths such as `schema.table.column` can encode sufficient regularity to support accurate lineage inference. These paths are widely available and consistently present across systems, even when deeper data instrumentation is lacking.

To illustrate this, consider SAP Business Warehouse, which defines standard tables reused across deployments. One such table, `KNA1`, contains columns like `KUNNR` (customer number), `ORT01` (city), and `MANDT` (client ID). These identifiers are stable and well-defined in vendor documentation. However, these fields may appear under very different names in downstream systems. A target system may refer to customer numbers as `customer_id` or `custnum`, or may rename cities as `location` or `city_name`.

Similar challenges of limited data access are addressed by Liu et al. [24], who introduce GRAM, a generative retrieval augmented matching framework that enables schema matching with minimal access to sensitive customer data using zero-shot and few-shot learning. In our work, we also restrict ourselves to using only accessible metadata, which makes our approach applicable in environments with strict data governance constraints. Moreover, our method can be deployed in a zero-shot setting to predict lineage links even in organizations where direct data access is not possible, which aligns with the broader trend of balancing data accessibility with regulatory and security concerns.

In some cases, profiling tools may be used to support lineage inference by performing classification tasks, such as identifying data types or semantic roles (e.g., detecting birthdates or postal codes), and by generating summary statistics, such as value distributions and aggregate statistics. While such signals can be valuable, they are not always available—particularly in federated environments or systems without integrated data catalogs. This further motivates our reliance on schema metadata as a primary source of signal.

To support inference in such scenarios, we employ models capable of capturing the latent semantics of names. In contrast to traditional schema matching, our goal is not to find symmetric equivalences, but to detect directional lineage relationships—even when names differ substantially or offer no obvious textual match.

Formally, given two datasets, *source* and *target*, each composed of sets of tables $S = \{S_1, S_2, \ldots, S_n\}$ and $T = \{T_1, T_2, \ldots, T_m\}$, where each table contains multiple columns $S_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$, the task is to identify lineage links of the form (s_{i_p}, t_{j_q}) indicating that column $s_{i_p} \in S_i$ is the source of $t_{j_q} \in T_j$.

In this work, we focus on single-source lineage relationships, where each target column is derived from exactly one source column. Although in principle some transformations may involve multiple inputs (multi-source lineage), 95.3% of observed lineage mappings in our real-world dataset follow a single-source pattern. This restriction both simplifies the modeling task and reflects the operational structure of many ETL pipelines, where columns are often directly forwarded, renamed, or minimally transformed.

While our approach is inspired by schema matching techniques, the task differs significantly in nature. Schema matching seeks symmetric semantic equivalence between columns or tables, often with the goal of integration or alignment. In contrast, lineage inference is directional and asymmetric: the question is not whether two elements mean the same thing, but whether one is derived from the other. For example, a column named unitsSold in one table might correspond to a column named totalUnits in another, with the latter derived through an aggregation operation such as summation. Such lineage relationships often exist despite differences in naming, requiring models to infer latent semantic links that go beyond surface-level similarity.

This challenge is further amplified by the structure of the task itself. It is characterized by extreme class imbalance: among the vast number of possible source-target column pairs, only a small fraction correspond to actual lineage. At the same time, many incorrect candidates exhibit surface-level similarity due to naming conventions or abbreviations, making them deceptively plausible. The core difficulty is therefore not finding connections between dissimilar elements, but rejecting misleading matches that appear similar on the surface.

This modeling difficulty is aligned with practical expectations. In many enterprise applications—such as compliance reporting, data audits, or automated documentation—users place greater value on the reliability of inferred links than on exhaustive coverage. This reinforces the importance of resolving the ambiguity introduced by similar-looking but unrelated columns.

4 RELATED WORK

4.1 Schema Matching

Schema matching is a foundational problem in data integration, focused on identifying semantically equivalent elements across heterogeneous schemas. Classic surveys [37, 42] classify approaches based on the use of metadata such as column names, data types, schema structure, and domain-specific dictionaries or ontologies. These early methods relied heavily on rule-based heuristics and string similarity metrics.

With the rise of machine learning, schema matching has increasingly been framed as a binary classification problem over column pairs. Approaches such as DeepMatcher [28] demonstrated that deep neural networks, including recurrent and attention-based architectures, can effectively match even noisy or partially formatted data. Subsequent work explored BiLSTM models over column names and descriptions [27], enabling better contextual understanding of naming patterns.

More recent models leverage transformer-based language representations such as BERT. For example, Zhang et al. [50] showed that pre-trained models can generalize well even in low-resource settings. The SMAT architecture [49], inspired by attention-over-attention mechanisms from question answering, enables schema alignment using only textual features, without access to instance-level data—making it suitable for privacy-sensitive domains such as healthcare.

Hybrid approaches have also gained traction. Asif et al. [1] combine heuristic rules with learned models to balance interpretability and robustness in specialized domains. Industry-focused studies, such as [29], further demonstrate the value of semantic schema alignment in commercial applications like product lifecycle management.

Recent work has also investigated large language models (LLMs) for schema matching [25, 31]. While these models achieve strong performance, they remain costly to apply at scale. Moreover, adapting LLMs to specialized enterprise schemas—where column names are often domain-specific and abbreviated—is nontrivial.

In contrast, our approach focuses on learning semantic similarity from schema metadata using scalable, embedding-based models. Rather than aiming for symmetric equivalence, as in most schema matching settings, we target directional lineage inference under class imbalance, where the goal is to detect whether one column is derived from another.

4.2 Reducing Search Space: Blocking and Filtering

In the context of schema matching, reducing the search space is a fundamental step for enabling scalable processing over large datasets. Naively comparing all possible pairs of schema elements results in a quadratic number of comparisons, which quickly becomes infeasible as the number of tables or columns increases. This brute-force approach is computationally prohibitive and unsuitable for real-world deployments, where schema catalogs may include thousands or even millions of elements.

To address this, modern systems incorporate candidate pruning strategies that aim to discard clearly irrelevant pairs early in the pipeline. These techniques are designed to retain only the most plausible candidate matches for further processing, thereby reducing both runtime and memory requirements. Efficient search space reduction not only enables matching to complete within practical time bounds, but also improves overall effectiveness by focusing model attention on meaningful comparisons.

Among the most widely used approaches are blocking and filtering techniques. As early as [2], it was shown that even simple similarity metrics, such as Jaccard or cosine similarity, are insufficiently scalable without aggressive candidate pruning. More recent work has emphasized blocking-based strategies, which group elements into smaller buckets that can be compared more efficiently.

Blocking techniques partition the candidate space using lexical or statistical heuristics. For example, Li et al. [22] propose simple token-based filtering: only elements sharing a common token are compared. Other strategies use TF-IDF to rank tokens and create compact candidate blocks, as in Sparkly [32]. Such approaches, while lightweight, often rely on surface-level features.

Filtering techniques use a similarity function and a threshold to directly prune unlikely candidates. Though potentially more accurate, they require more computation. Papadakis et al. [30] review such methods and note that scalability remains an issue without further optimization.

Recent work has explored the use of deep learning for prefiltering. Thirumuruganathan et al. [46] benchmark attention-based and embedding-based models, demonstrating that neural networks can be effective not only for matching, but also for filtering.

In our approach, we use a bi-encoder architecture to efficiently filter column pairs based on learned semantic similarity. This enables

high-throughput candidate scoring using GPUs, while preserving precision and flexibility across datasets. Unlike traditional blocking, our method supports fine-grained, model-driven filtering without requiring manual token rules or thresholds.

4.3 Hard Negative Mining

Hard negative mining is a key technique in dense retrieval for training models that can distinguish between semantically similar but incorrect matches. Unlike easy negatives, which are clearly unrelated to the query, hard negatives are misleadingly close in meaning and force the model to make finer distinctions.

Dense Passage Retrieval (DPR) [19] introduced in-batch negatives, treating all non-positive examples in the same mini-batch as negatives. While efficient, many of these are too easy to be effective training signals.

To address this, ANCE [48] proposed retrieving nearest-neighbor negatives from the corpus using the current model, selecting only the most semantically similar yet incorrect candidates. RocketQA [36] refined this by filtering out false negatives—documents that might accidentally contain the correct answer.

ColBERT [20], while not explicitly using hard negatives, relies on token-level interactions to force more precise matching. A recent survey [51] emphasizes that semantically close negatives are essential for training accurate retrievers.

We adopt this principle by constructing training sets with hard negatives: column pairs that are lexically or semantically similar but do not represent true lineage. This improves the model's ability to filter misleading matches and focus on high-confidence links.

5 LINEAGE INFERENCE UNDER PRACTICAL CONSTRAINTS

While our task shares surface-level similarities with schema matching, the modeling requirements are substantially different. Traditional matching systems often rely on semantic similarity between column names or descriptions to identify equivalences. In contrast, lineage inference is an asymmetric task: the goal is to determine whether a target column is derived from a specific source column. This distinction shapes both how we formulate the task and how we train models to solve it.

A key challenge is the nature of real-world schema names. Database columns are frequently named using nonstandard abbreviations, inconsistent casing, and domain-specific shorthand. For example, a column representing a customer purchase transaction may appear as `CustPurchTran`, `CusPurTrans`, `CPT`, or `CustPT`, often without delimiters or clear token boundaries. Such naming practices limit the effectiveness of pre-trained language models, which are tuned to natural text with regular structure.

Fig. 2 illustrates a real-world example of source and target schemas in column-level lineage inference, where different naming patterns appear across systems.

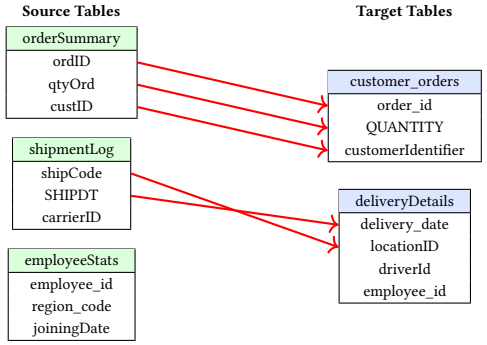


Figure 2: Illustrative source and target tables with column names in different naming styles (e.g., camelCase, snake_case, uppercase, compact acronyms).

To handle large-scale industrial catalogs—where a single schema may contain tens of thousands of columns—we require a filtering method that is both scalable and capable of learning domain-specific patterns. Prior work on candidate pruning has largely relied on heuristic techniques or rule-based blocking, which can struggle in high-similarity settings where many column names appear plausible but are not actually related.

We therefore adopt a learned filtering stage based on a bi-encoder architecture. This enables the model to capture dataset-specific similarities while supporting highly parallelized inference on GPUs. By encoding source and target columns independently, we achieve efficient computation across quadratic number of candidate pairs. Compared to heuristic filters, our bi-encoder substantially reduces the number of pairs passed to the next stage, while preserving most valid lineage links.

This reduction makes it feasible to apply a second-stage model with greater expressive power. We use a cross-encoder to jointly encode candidate pairs and classify them as lineage or not. While slower than the bi-encoder, the cross-encoder is significantly more precise, especially in rejecting high-similarity distractors that would otherwise inflate false positive rates.

The two-stage architecture also offers practical modeling advantages. In particular, it allows us to control which examples are shown to the cross-encoder, enabling targeted use of hard negative mining. Given the extreme class imbalance in our task, where only a small fraction of pairs represent true lineage, such negatives are essential for training robust discriminators. Our experiments confirm that this setup consistently achieves strong performance while maintaining tractability across large-scale real-world datasets.

We describe the architecture in more detail, along with a visual summary, in Section 6.2.

6 MODELING SEMANTIC SIMILARITY

6.1 From Word Embeddings to Transformers

Word embeddings have become a cornerstone of natural language processing, providing powerful representations of words in continuous vector spaces. Notable methods include Word2Vec [26], which uses shallow neural networks to learn distributed word representations based on co-occurrence patterns, capturing semantic relationships. GloVe [33] combines local context with global co-occurrence statistics to create embeddings that reflect both syntactic and semantic information. FastText [3], an extension of Word2Vec,

incorporates subword information, allowing for the representation of out-of-vocabulary words and morphological variations.

Attention mechanisms represent a significant advancement in natural language processing [10], complementing rather than replacing word embeddings. While embeddings like Word2Vec, GloVe, and FastText capture semantic relationships in a fixed-dimensional space, attention mechanisms dynamically weigh the importance of words in a sequence. Initially popularized in sequence-to-sequence models for machine translation, attention allows models to focus on relevant parts of the input, alleviating issues with fixed context windows and improving long-range dependencies. This context awareness enhances both language understanding and generation, making attention mechanisms more flexible and adaptive for sequential data, leading to notable improvements in various NLP tasks.

Transformers, introduced in the groundbreaking paper "Attention is All You Need" [47], rely entirely on self-attention, enabling parallel processing of sequences and handling long-range dependencies effectively. These architectures have become a foundation for many modern NLP systems, particularly in tasks involving sequence modeling, classification, and semantic comparison.

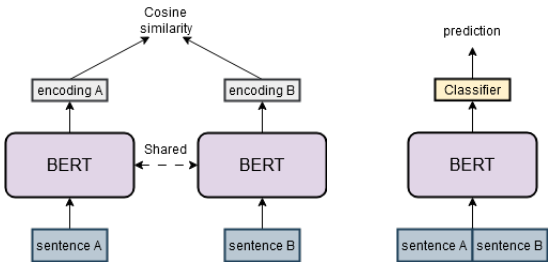
In our work, we leverage transformer-based encoders (specifically bi-encoder and cross-encoder variants) to capture semantic relationships between schema elements. Unlike traditional NLP tasks, our inputs are structured column identifiers, often composed of domain-specific shorthand and compact naming patterns rather than natural text, requiring models that are robust to abbreviations and naming inconsistencies.

6.2 Semantic Feature-Comparison Model

The Semantic Feature-Comparison Model is designed to evaluate the semantic similarity between two textual inputs. It transforms each input into a representation that captures its underlying meaning, enabling effective comparison. This model is helpful in tasks such as textual similarity, semantic search, and understanding relationships between texts.

Bi-encoder architecture independently encodes each input, producing two separate representations, as presented in Sentence-BERT [40]. These representations are then compared using a distance metric (e.g., cosine similarity). The bi-encoder is computationally efficient, making it ideal for large-scale tasks where speed and scalability are crucial, such as in real-time applications or systems handling large datasets. Since the inputs are encoded separately, the model does not capture interdependencies between them as effectively as other methods.

Cross-encoder architecture, on the other hand, processes both inputs together as a pair, encoding them jointly. This allows the model to capture more complex, context-dependent relationships between the inputs. BERT [7] is an example of such a model, commonly used in a cross-encoder setup. However, this approach is more computationally expensive, as it requires processing the entire pair for each comparison. The *cross-encoder* is more accurate and is particularly useful in tasks where precision is crucial, such as question answering or tasks that involve detailed sentence pair classification.



(a) Bi-encoder **(b) Cross-encoder**

Figure 3: Comparison of Bi-encoder and Cross-encoder

.

The primary difference between the two approaches lies in their efficiency and the level of contextual understanding they offer. While the *bi-encoder* is faster and more scalable, the *cross-encoder* provides a more nuanced and precise evaluation of the relationship between inputs by considering them together in context. The differences between these architectures are illustrated in Fig. 3, which highlights how each method processes inputs and compares their representations.

In the next section, we show how these semantic similarity mechanisms are integrated into our complete two-stage inference architecture, including training procedures and data handling.

7 MODEL

7.1 General Overview

Figure 4 presents the overall architecture of our inference pipeline. The process begins by segmenting both source and target schemas, followed by a two-stage pipeline comprising filtering and classification.

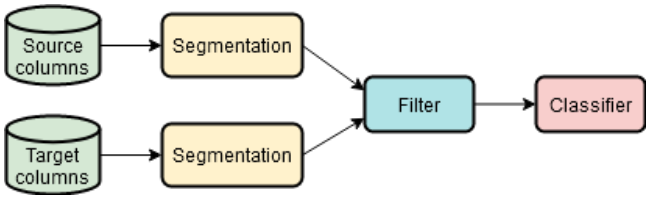


Figure 4: General overview of the proposed approach

Filtering is performed using a bi-encoder model, which computes embeddings for source and target columns independently. This allows for efficient candidate scoring across all column pairs using GPU acceleration. The filtered pairs are then passed to a cross-encoder, which jointly encodes each candidate pair and makes the final prediction.

The cross-encoder architecture does not assume symmetry between inputs, making it well-suited to the directional nature of data lineage inference. The architectural differences between bi- and cross-encoders are discussed in Section 6.2.

A detailed explanation of the segmentation logic and training procedure for both models follows in the subsequent sections.

7.2 Schema Name Segmentation

To preprocess column names for modeling, we first segment them into interpretable subunits. Schema elements in real-world databases often use compressed naming conventions, such as concatenated abbreviations (e.g., `CustPurchTran`, `CPT`), camelCase, or domain-specific acronyms. These forms can obscure semantic content, especially for pre-trained models expecting natural language input.

We apply a statistical segmentation algorithm trained on schema labels that include natural delimiters (e.g., underscores, camelCase, or hyphens). These serve as weak supervision to estimate n-gram frequencies over token sequences.

Given an input character sequence, we consider all possible segmentations into word sequences. We denote a candidate segmentation as $w = (w_1, w_2, \ldots, w_n)$, and our goal is to select the most plausible one. Additionally, for notational clarity, we define the subsequence w_i^j to denote $(w_i, w_{i+1}, \ldots, w_j)$ for any $i \leq j$.

To estimate the quality of each segmentation, we approximate the probability of the entire sequence using an n-gram language model:

$$P(w_1, w_2, \ldots, w_n) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-k+1}^{i-1}),$$

for a fixed value of k. Here, w_{i-k+1}^{i-1} denotes the $(k-1)$-length prefix context preceding token w_i, with the convention that for positions where $i < k$, we instead use the prefix w_1^{i-1}. This allows the model to gracefully handle sequence boundaries without requiring explicit start-of-sequence tokens. The conditional probabilities $P(w_i \mid \cdot)$ are estimated from training data using observed frequencies $f(\cdot)$.

We adopt the *Stupid Backoff* scoring scheme [5], which recursively backs off to shorter contexts with a fixed decay factor α (typically 0.4):

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{f(w_{i-k+1}^{i})}{f(w_{i-k+1}^{i-1})}, & \text{if } f(w_{i-k+1}^{i}) > 0 \\ \alpha \cdot S(w_i \mid w_{i-k+2}^{i-1}), & \text{otherwise} \end{cases}$$

Here, $f(\cdot)$ denotes the raw frequency of an n-gram in the training corpus. The total segmentation score $S(w)$ is computed for each candidate split, and we select the segmentation that maximizes the sum of log scores over all segments.

To find the best segmentation, we evaluate all valid segmentations of the input sequence. Since the number of possible splits grows exponentially with sequence length, we implement dynamic programming with memoization to avoid redundant computation and ensure efficient inference.

To handle unknown words not present in the training data, we apply a fallback heuristic that penalizes longer unobserved tokens:

$$S(w_i) = \frac{C}{T \cdot b^{|w_i|}},$$

where T is the total number of unigrams, C is a smoothing constant, and $|w_i|$ is the token length.

The main advantage of this approach is its simplicity and speed—the method requires no sophisticated learning or advanced machine learning models. Despite its simplicity, it is effective in real-world scenarios and can be easily adapted to naming conventions specific to a given environment. Ultimately, we use only a bigram model to evaluate segmentation quality, which is sufficient for the short names typical of database schemas, where context beyond a single neighboring token usually adds little additional information. Moreover, limiting the model to bigrams keeps computational complexity very low, allowing for fast processing at scale.

7.3 Model Training

While bi-encoders and cross-encoders ultimately aim to assess semantic relationships, their architectural differences lead to distinct roles during training and inference. Prior work has explored hybrid strategies that combine both models: in [6, 45], the more accurate cross-encoder is used to genterprise systems. Despite its simplicity, the segmenter is highly effective in practice and robust to the kinds of inconsistencies commonly found in schema names. A bigram model is sufficient for most cases, as column names tend to be short and local context dominates. Late high-quality synthetic labels for the bi-encoder. This is especially beneficial when the available dataset lacks full coverage of positive examples.

In our architecture, we reverse this flow: the bi-encoder serves as a fast, learned filter. Rather than making definitive predictions, it eliminates clearly implausible candidates while preserving nearly all true lineage links. This pre-filtering dramatically reduces the number of column pairs considered by the more computationally expensive cross-encoder. At this stage, minimitzing false negatives is more important than maximizing precision.

An overview of the training pipeline is presented in Fig. 5.

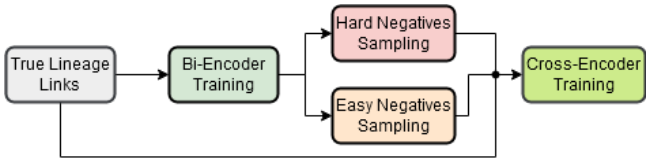


Figure 5: Training pipeline

7.3.1 Entry Encoding. Language models operate on sequences of tokens, so we convert column path names into token sequences to make them suitable for processing by the network. Although the metadata includes host, database, and schema names, we omit these components during encoding, as they are repetitive and offer limited discriminative value for the task. Instead, we focus on table and column names, which carry higher informational content. These are separated using the special `[SEP]` token, as shown in Equation 1, which BERT uses in the next sentence prediction task to mark the boundary between segments. Moreover, the values for host, database, and schema are identical for both source and target entries, as described in Subsection 7.4.

$$entry(path) = \texttt{<table name>}\ [SEP]\ \texttt{<column name>} \quad (1)$$

7.3.2 Bi-encoder. The bi-encoder architecture is trained using the Multiple Negatives Ranking (MNR) loss, which is commonly employed in retrieval and ranking tasks. This loss function encourages the model to learn vector representations of source and target entities (i.e., table and column names) such that semantically similar

entities are embedded close to each other in the representation space. During training, the bi-encoder processes a mini-batch of size n, treating all non-matching pairs within the batch as negatives, enabling the model to learn from multiple examples concurrently and significantly accelerating training.

A key advantage of using MNR in this context is its scalability: the bi-encoder computes embeddings independently for each entity, enabling efficient similarity estimation at inference time. This makes it an effective filtering mechanism, capable of narrowing down the candidate set before passing pairs to the more precise cross-encoder.

Furthermore, because the bi-encoder is trained to capture general similarity patterns rather than make final matching decisions, it is particularly well suited for hard negative mining. It helps identify pairs that appear similar but are not actually linked, providing challenging cases for the cross-encoder to refine its discrimination capability.

7.3.3 Cross-encoder. The cross-encoder is trained as a classifier designed to make final decisions about whether a given pair of columns is semantically related. Unlike the bi-encoder, which processes entities independently and assumes symmetry in the data, the cross-encoder jointly encodes both elements of the pair. This allows the model to capture full contextual interactions and token-level dependencies, enabling it to identify subtle distinctions between similar-looking but semantically unrelated columns. As a result, the cross-encoder can detect more complex relationships that may be missed by the bi-encoder.

An alternative approach to combining bi-encoder and cross-encoder models was proposed in [23], where both components are used concurrently at prediction time. In their method, key sentences are first extracted based on entity presence to reduce noise and input length. The bi-encoder computes global embeddings, while the cross-encoder performs fine-grained token-level matching on the pruned content. The final decision is made by fusing signals from both models. In contrast, our approach uses the bi-encoder strictly as a filtering and hard negative mining component, while the cross-encoder is reserved for final classification, operating only on the candidate pairs selected in the earlier stage.

7.3.4 Hard Negative Mining. Unlike common approaches where hard negatives are primarily used to improve encoder performance, in our method they serve a different purpose. The goal is not to maximize the precision of the bi-encoder itself, but to use its similarity estimates as a difficulty signal for constructing a more effective training set for the cross-encoder.

The bi-encoder acts as a preliminary selector: it estimates the similarity between source and target columns and identify negative examples that are deceptively plausible. As a result, the training set for the cross-encoder includes a higher proportion of such challenging cases, which would be rare under uniform sampling. This enables the model to learn from more realistic and demanding scenarios—ones that even strong semantic encoders may struggle with.

An alternative strategy was proposed in [8], where hard negatives for bi-encoder training were selected using a cross-encoder. While this approach yields highly precise training examples, it requires prior access to a well-tuned reference model, which may not always be feasible. In contrast, our pipeline is stage-based: the bi-encoder is trained independently, and only then used to estimate difficulty and construct training batches for the cross-encoder.

A key advantage of this strategy is that all training examples, positive and negative, come from the same organization. This mirrors the actual use case, where the model must discriminate between valid and invalid lineage links within a single enterprise schema. Consequently, it is forced to learn domain-specific patterns, naming conventions, and abbreviations that are consistent across all examples. The model cannot rely on stylistic cues but must learn the underlying semantic distinctions.

This strategy is simple to implement and integrates seamlessly into the pipeline. Since the bi-encoder already computes embeddings during inference, it can be reused for difficulty estimation without introducing new models or requiring manual labeling.

The effectiveness of cross-encoder training also depends on the composition of the training set, particularly the balance between positive, easy negative, and hard negative examples. In our setup, we adopt a 1:1:1 ratio, providing the model with an equal number of positives, hard negatives, and easy negatives. This balanced sampling improves the model's robustness and generalization, exposing it to a diverse range of decision boundaries.

A similar observation was made in [6], which systematically studied the effect of sampling strategies on downstream classification performance. Further details about dataset construction and schema organization are provided in subsection 7.4.

7.4 Evaluation

In this study, we work with multiple schema pairs, each consisting of a source and a target schema, often containing several thousand columns. Unlike traditional schema matching, which typically compares complete schemas or databases, our approach operates over a large number of such pairs, with each pair treated independently. This setup reflects real-world scenarios in which lineage must be reconstructed between two provided schemas, without global visibility.

Given the scale of the dataset, we follow established practices to split it into training and test sets (90/10) in a way that supports reliable evaluation. However, due to the presence of structural and organizational dependencies between some schemas, care must be taken to avoid data leakage. Even minimal overlap between training and test data can lead to an unrealistic assessment of performance.

To prevent this, we enforce a strict separation policy: all schema pairs associated with a given company are assigned exclusively to either the training or the test set, but never both. This ensures that the model is not indirectly exposed to test-time patterns during training.

This strategy guarantees that the test set represents entirely unseen data, enabling an unbiased evaluation of the model's ability to generalize to new organizational domains and naming conventions. This split strategy forms the foundation for the experiments described in Section 8.

8 EXPERIMENTS

8.1 Dataset

Our experiments are grounded in a diverse collection of datasets sourced from thousands of real-world enterprise projects, including data from large, reputable organizations such as banks, insurance providers, and industrial companies. These datasets encompass internal repositories and databases that reflect authentic production systems. In total, the data originates from 5,673 distinct sources and includes 77,237 schemas. Each schema represents a logical unit that groups tables and columns for matching purposes.

The dataset comprises over 47 million columns, with each schema containing an average of 609 columns. However, the distribution varies considerably, with a standard deviation of 4,360 columns per schema, and some of the largest schemas containing more than 680,000 columns.

Our experimental setup involves 57,546 schema pairs for which we predict lineage connections. On average, each schema pair yields nearly 26 million possible column pairs, with some pairs generating over 460 billion. Across the entire dataset, there are approximately 16 million validated lineage links, which translates to an average of 284 links per schema pair. Consequently, there is roughly one positive link for every 90,000 possible pairs, highlighting the highly imbalanced nature of the problem and underscoring the critical importance of effectively rejecting false positives to achieve strong overall performance.

8.1.1 Limitations. Our study focuses on data lineage inference in real-world production environments. The original datasets used in our experiments are derived from enterprise systems and contain sensitive structural metadata that reflects internal data flows and business logic. Releasing such information would pose significant confidentiality and security risks, as it could expose core elements of organizational operations, system design, and data governance practices.

To support transparency and understanding of our approach, we release a synthetic dataset that mirrors the structure, scale, and statistical properties of the original data while ensuring no sensitive information is included. This synthetic data allows readers to examine the model input format and evaluation protocol, though it does not support full reproducibility. We believe this compromise strikes an appropriate balance between methodological clarity and the obligation to protect confidential infrastructure.

To illustrate the structure of the dataset used in our experiments, we include two supplementary files: one listing all tables and columns, and another containing all lineage links, where both the source and target columns originate from the same schema.

8.2 Compared Algorithms

For comparison, we implemented two recently published algorithms designed for schema or entity matching:

- **DITTO** [21] — An entity matching model that differs from our approach by employing blocking, which requires matching entries to share at least one word in common. Additionally, it uses a different method for generating hard examples.
- **AI Match** [14] — A model that first performs indirect filtering by matching tables and then predicts column-level matches within those tables.

To ensure a fair comparison, we evaluated all methods on segmented data, thereby standardizing the experimental conditions and eliminating discrepancies in data preprocessing that could otherwise influence performance outcomes.

8.3 Experiment Setup

In this study, we trained our model on a machine equipped with two NVIDIA A100 GPUs, providing the necessary computational power to handle large-scale datasets and complex tasks efficiently.

We evaluated the model using two main approaches: within-project and cross-project. In the within-project approach, we trained the model on a subset of data from a single company and evaluated it on the remaining data from the same company. In contrast, the cross-project approach involved training the model on data from multiple companies and testing it on data from a completely different company. Since the cross-project setting more accurately reflects real-world scenarios, we prioritized it in our evaluations and in our comparisons with other methods (see subsection 8.2).

To better understand the differences between these approaches, we compared the performance of models trained in both within-project and cross-project settings for the three largest companies in our dataset. We ensured that each company's data was organized into distinct schemas to facilitate experiments closely resembling real-world conditions. Additionally, we conducted an ablation study to assess the impact of segmentation and hard negative mining on model performance.

For our task, a single example corresponds to a pair of schemas (see subsection 7.4). For each schema pair, we inferred the data lineage links predicted by our model and compared them with the ground-truth links. Since a single example can involve an enormous number of potential links—sometimes requiring tens of thousands of network queries—we also measured the time taken for each query to provide a comprehensive evaluation of the model's efficiency.

To accelerate training and enhance model performance, we fine-tuned our model using pre-trained language models based on the BERT architecture.

8.4 Results

To evaluate model performance, we employed four widely recognized classification metrics: precision, recall, F1 score, and PR-AUC (Precision-Recall Area Under Curve). The use of PR-AUC instead of ROC-AUC was motivated by the highly imbalanced nature of the dataset, which favors precision-recall analysis. For each model, we selected the classification threshold that maximizes the F1 score.

A single training session lasted 10 hours for the bi-encoder and 4 hours for the cross-encoder. Notably, a significant portion of the overall pipeline runtime—approximately 40 hours—was dedicated to hard negative mining. This process involved generating all possible source-target pairs for each set of schemas and evaluating their difficulty using the bi-encoder model.

8.4.1 Main results. We conducted experiments in a manner that better reflects real-world scenarios: training the model on data from parts of the companies and testing it on different ones. We

Table 1: Performance measures for all models

Model	Precision	Recall	F1	AUC	time
Bi+Cross-Encoder	0.73	**0.81**	**0.77**	**0.80**	7.5s
DITTO	0.66	0.77	0.71	0.70	503s
AI Match	**0.75**	0.33	0.46	0.27	1.6s

Table 2: Ablation study

Model	Precision	Recall	F1	AUC
Full Model	**0.73**	**0.81**	**0.77**	**0.80**
w/o Segmentation	0.56	0.76	0.65	0.65
w/o Hard Negatives	0.65	0.79	0.71	0.61

then compared the performance of our approach with the models presented in subsection 8.2. Precision-recall curves are shown in Fig. 6, and the numerical values of the evaluated metrics are provided in Table 1.

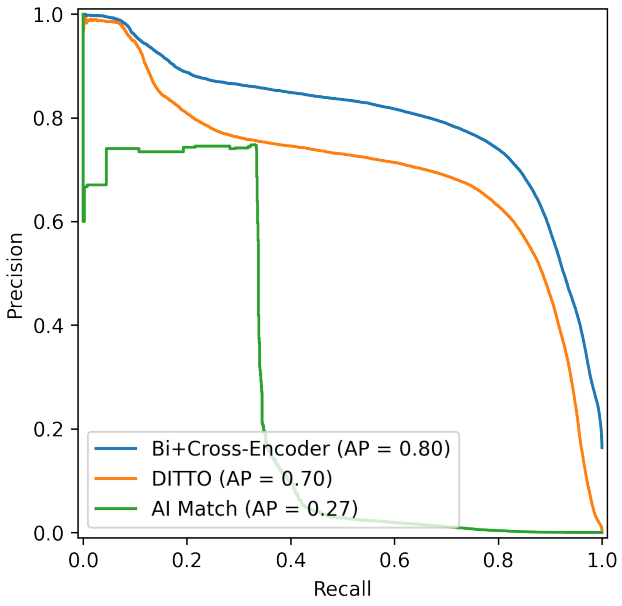


Figure 6: Precision-Recall curves for all models (AP = average precision = PR-AUC)

Our approach outperformed the DITTO model in terms of F1 and AUC while being significantly faster due to its effective candidate filtering. Our filter reduces the number of candidates by approximately a thousandfold, discarding only about 2% of true links. By comparison, token-sharing filters reduce candidates by only tenfold, with a similar number of true links rejected. Additionally, the bi-encoder enables more effective hard negative mining, leading to better overall performance.

The AI Match model achieved lower scores, mainly because it uses a pre-trained model that is not well-suited to this specific problem. As illustrated in Fig. 6, achieving a recall higher than 0.33 with AI Match requires substantially reducing precision. Consequently, its maximum F1 score is reached at a threshold close to 1, meaning when the source and target names are nearly identical.

Based on the results, it is clear that in the case of such a heavily imbalanced problem, the key to success lies in the effective reduction of false positives. Without this, achieving a satisfactory precision—and thus a comprehensive performance—is very challenging. Our model, by directly optimizing the filtering step based on the data, clearly outperforms the alternatives.

8.4.2 Role of bi- and cross-encoders in the architecture. To complement the main results, we compared the performance of the cross-encoder and bi-encoder. Most studies in this area rely on heuristic filters and a single machine learning model (typically similar to a bi-encoder) to determine matching. This raises a natural question: what is the benefit of using two models, and did they actually learn something different, with each fulfilling its own role?

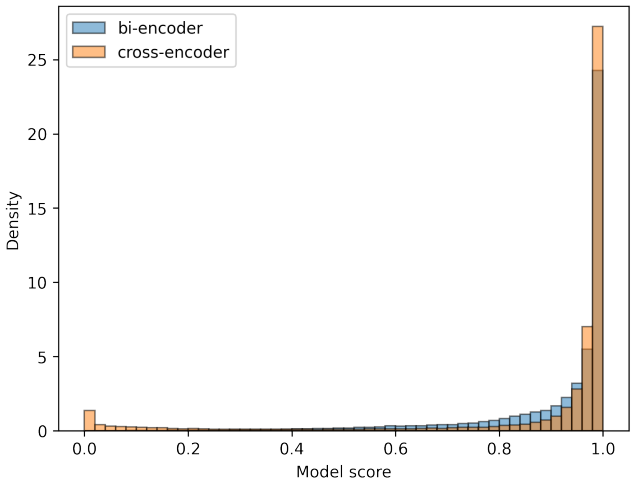
Figure 7 shows the distribution of model predictions from both of our trained models. For non-link examples, we included only those with a bi-encoder similarity of at least 0.5 to improve readability, since values below this threshold are much more numerous and would otherwise dominate the distribution, obscuring meaningful comparisons.

Their prediction distributions differ substantially. The cross-encoder is significantly better at rejecting false positives—which, as previously discussed, is crucial for this problem—while only slightly increasing the number of rejected true links (false negatives).

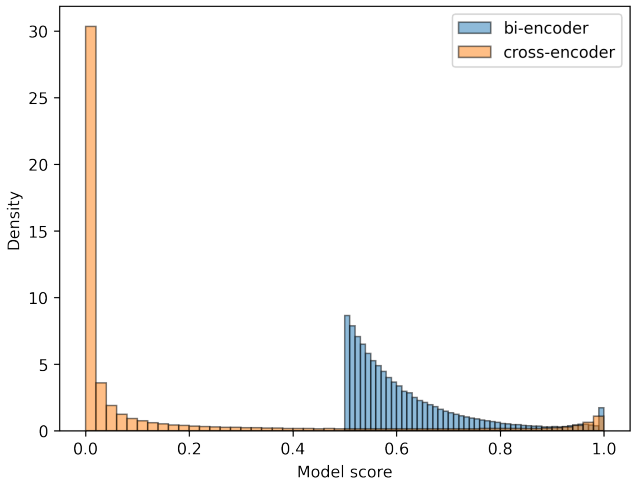
The distribution for true links is also interesting: the cross-encoder more often returns boundary values, whereas the bi-encoder more frequently assigns intermediate values. Ultimately, both models effectively fulfill their roles, demonstrating that the bi-encoder + cross-encoder architecture is well-justified and difficult to replace with a single model.

8.4.3 Ablation study. We compared our complete model with a version where two key components were removed. The first component is segmentation, as described in Subsection 7.2. In the alternative approach, we trained both models directly on the column names appearing in the tables. The second component is the generation of hard examples: instead, we trained a model using randomly selected negative examples, maintaining a 1:1 ratio of negatives to positives. The results of this comparison are presented in Fig. 8 and Table 2.

The results demonstrate that both components have a significant impact on overall performance. Both ablated models exhibited an optimal threshold above 0.99, indicating difficulty in distinguishing hard negatives. The model trained without hard negative mining struggled even with examples it was most confident about, leading to a steep drop in performance in Fig. 8. Interestingly, despite its considerably worse overall performance, the model without segmentation identified more positive examples at the same threshold as the model with segmentation. This suggests that the way source and target column names are written might itself serve as a useful indicator.



(a) Prediction distribution for true links



(b) Prediction distribution for non-links

Figure 7: Comparison of bi- and cross-encoder prediction distributions for true links and non-links

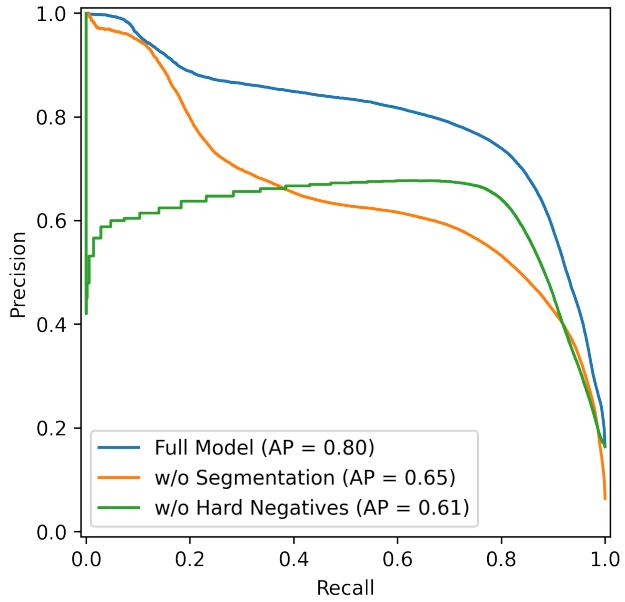


Figure 8: Impact of different model components (ablation study)

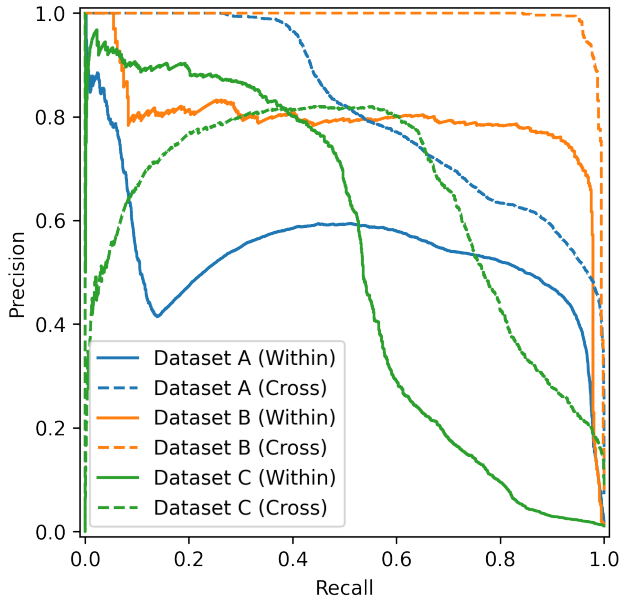


Figure 9: Comparison of cross-project and within-project evaluation

8.4.4 Cross/within-project comparison. We conducted additional experiments by training models on subsets of schema pairs from the three largest companies (anonymized as A, B, and C). Additionally, we retrained our model after removing all data from these three datasets to compare performance. The results are presented in Table 3 and Figure 9.

The cross-project approach consistently outperformed the within-project approach in all three datasets. At first glance, this might seem surprising. However, the within-project scenario involves predicting data lineage within a schema using information from different parts of the same organization. This allows us to capture conventions specific to that company, although these conventions may vary considerably across departments in large organizations. On the other hand, different companies often use similar tools and follow comparable practices when designing their schemas. By leveraging data from other companies, the cross-project model can learn a broader range of patterns that characterize how schemas

Table 3: Comparison cross/within-project approaches

Dataset	Method	Precision	Recall	F1	AUC
A	Cross-project	**0.62**	**0.87**	**0.72**	**0.82**
	Within-project	0.52	0.82	0.63	0.54
B	Cross-project	**0.99**	**0.95**	**0.97**	**0.99**
	Within-project	0.76	0.92	0.83	0.79
C	Cross-project	**0.77**	**0.64**	**0.70**	**0.63**
	Within-project	0.71	0.49	0.58	0.52

typically evolve, which ultimately leads to significantly better results.

8.5 Threats to validity

8.5.1 Internal validity. The model relies on the path of data artifacts, using only the names of tables and columns as input. While this approach limits the data to easily accessible database features, it is important to note that restricting the model to this form of input can impact its performance. Expanding the input to include additional metadata or other data sources might improve the model's effectiveness. However, as highlighted in Section 1, the core idea is to work with minimal, readily available information related to the "linguistic" aspects of schema matching, especially when direct access to production data may not be feasible.

Despite careful dataset preparation, there is always a possibility that the available data does not fully reflect real-world conditions. In some cases, access to procedures or other relevant database elements may be limited, which could result in incomplete lineage information. Moreover, databases change over time—schemas are modified, tables are added or removed, and naming conventions evolve. As a result, the reconstructed lineage may not accurately reflect the historical state of the database at the time the processes were originally implemented.

8.5.2 External validity. Although we used a relatively large dataset for this study, it primarily consists of data from the ETL processes of financial institutions. This could introduce biases in the naming conventions for databases, tables, and columns. As a result, the model's effectiveness may not be fully transferable to other business domains or specific datasets.

9 CONCLUSIONS AND FUTURE WORK

Data lineage has demonstrated its essential role in ensuring the quality, transparency, and reliability of data within modern information systems. By systematically documenting the flow of data from its sources through various transformations to its final outputs, organizations can gain invaluable insights into the origins, dependencies, and implications of their data assets. However, the tracking of data origins can sometimes be prevented or severely hampered by various circumstances.

In this work, we presented an approach to data lineage inference inspired by methods used in schema matching. While traditional schema matching often relies on heuristics such as the presence of common words or measures like TF/IDF, our approach distinguishes itself by utilizing a bi-encoder to estimate initial similarity. This network acts as a filter, enabling the generation of hard examples for further processing by a cross-encoder. The bi-encoder facilitates more targeted matching of data artifacts, thereby improving the overall performance of the inference process. Additionally, by focusing solely on the schema metadata for prediction, the model enhances its adaptability to different ecosystems.

The results from a real-world dataset demonstrate that this method, which combines a bi-encoder and cross-encoder, achieves promising accuracy using only the paths of the artifacts. However, restricting the approach to database metadata limits its potential. Previous studies on schema matching have shown that instance-based methods can outperform schema-based approaches in many cases. Given that direct access to data is often restricted due to sensitivity concerns in real-world applications of data lineage, future work could explore integrating data statistics—such as data profiling, already mentioned in Section 5—to further improve the model's performance.

Furthermore, although the cross-company training generally yielded stronger results overall, it remains important to consider company-specific characteristics. In practice, organizations often adopt not only distinct naming conventions but also unique ways of organizing data flows—such as the size and complexity of schemas, the number and nature of inter-table connections, and typical transformation patterns. These differences can lead to substantial variation in schema structures, as illustrated in Fig. 9. To address this, future research could explore incorporating organization-specific metadata or fine-tuning hyperparameters (such as similarity thresholds) to align the model more closely with the unique characteristics of how a particular enterprise manages its data flows.

ACKNOWLEDGMENTS

This work is the result of Research Project No. DWD/4/66/2020 supported and funded by the Ministry of Education and Science in Poland.

REFERENCES

[1] Md Asif-Ur-Rahman, Bayzid Ashik Hossain, Michael Bewong, Md Zahidul Islam, Yanchang Zhao, Jeremy Groves, and Rory Judith. 2023. A semi-automated hybrid schema matching framework for vegetation data integration. *Expert Systems with Applications* 229 (2023), 120405.

[2] Roberto J Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*. 131–140.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the association for computational linguistics* 5 (2017), 135–146.

[4] Shawn Bowers, Timothy McPhillips, and Bertram Ludäscher. 2012. Declarative rules for inferring fine-grained data provenance from scientific workflow execution traces. In *Provenance and Annotation of Data and Processes: 4th International Provenance and Annotation Workshop, IPAW 2012, Santa Barbara, CA, USA, June 19-21, 2012, Revised Selected Papers 4*. Springer, 82–96.

[5] Thorsten Brants, Ashok Popat, Peng Xu, Franz Josef Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 858–867.

[6] Yuxin Chen, Zongyang Ma, Ziqi Zhang, Zhongang Qi, Chunfeng Yuan, Bing Li, Junfu Pu, Ying Shan, Xiaojuan Qi, and Weiming Hu. 2024. How to Make Cross Encoder a Good Teacher for Efficient Image-Text Retrieval?. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 26994–27003.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill

Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186.

[8] Hande Dong, Jiayi Lin, Yanlin Wang, Yichong Leng, Jiawei Chen, and Yutao Xie. 2024. Improving Code Search with Hard Negative Sampling Based on Fine-tuning. In *2024 31st Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 221–230.

[9] Hao Fan. 2002. Tracing data lineage using automed schema transformation pathways. In *British National Conference on Databases*. Springer, 50–53.

[10] Andrea Galassi, Marco Lippi, and Paolo Torroni. 2020. Attention in natural language processing. *IEEE transactions on neural networks and learning systems* 32, 10 (2020), 4291–4308.

[11] Eduardo González López de Murillas, Hajo A Reijers, and Wil MP van der Aalst. 2017. Everything you always wanted to know about your process, but did not know how to ask. In *Business Process Management Workshops: BPM 2016 International Workshops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers 14*. Springer, 296–309.

[12] Philip J. Guo and Margo I. Seltzer. 2012. BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure. In *4th Workshop on the Theory and Practice of Provenance, TaPP'12, Boston, MA, USA, June 14-15, 2012*, Umut A. Acar and Todd J. Green (Eds.). USENIX Association.

[13] Alon Halevy, Flip Korn, Natalya F Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*. 795–806.

[14] Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. 2020. It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings. In *AIDB@VLDB 2020, 2nd International Workshop on Applied AI for Database Systems and Applications, Held with VLDB 2020, Monday, August 31, 2020, Online Event / Tokyo, Japan*, Bingsheng He, Berthold Reinwald, and Yingjun Wu (Eds.).

[15] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. 2017. A survey on provenance: What for? What form? What from? *The VLDB Journal* 26 (2017), 881–906.

[16] Felipe Alex Hofmann. 2020. *Tracer: a machine learning approach to data lineage*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[17] Robert Ikeda and Jennifer Widom. 2009. Data lineage: A survey. *Stanford University Publications. http://ilpubs. stanford. edu* 8090, 918 (2009), 1.

[18] Andrej Jurčo. 2023. *Data Lineage Analysis for PySpark and Python ORM Libraries*. Master's thesis. Univerzita Karlova, Matematicko-fyzikální fakulta.

[19] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering.. In *EMNLP (1)*. 6769–6781.

[20] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 39–48.

[21] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.* 14, 1 (sep 2020), 50–60.

[22] Yuliang Li, Jinfeng Li, Yoshi Suhara, AnHai Doan, and Wang-Chiew Tan. 2023. Effective entity matching with transformers. *The VLDB Journal* 32, 6 (2023), 1215–1235.

[23] Jianbo Liao, Mingyi Jia, Junwen Duan, and Jianxin Wang. 2023. FBC: fusing bi-encoder and cross-encoder for long-form text matching. In *ECAI 2023*. IOS Press, 1473–1480.

[24] Xuanqing Liu, Runhui Wang, Yang Song, and Luyang Kong. 2024. GRAM: Generative Retrieval Augmented Matching of Data Schemas in the Context of Data Security. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5476–5486.

[25] Yurong Liu, Eduardo Pena, Aecio Santos, Eden Wu, and Juliana Freire. 2024. Magneto: Combining Small and Large Language Models for Schema Matching. *arXiv preprint arXiv:2412.08194* (2024).

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[27] Amnai Mohamed, Choukri Ali, Youssef Fakhri, Gherabi Noreddine, et al. 2022. Schema Matching Based On Deep Learning Using LSTM Model. In *2022 IEEE 3rd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*. IEEE, 1–5.

[28] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*. 19–34.

[29] Hakju Oh, Albert Jones, Tim Finin, et al. 2023. Employing Word-Embedding for Schema Matching in Standard Lifecycle Management. *Journal of Industrial Information Integration* (2023), 100547.

[30] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)* 53, 2 (2020), 1–42.

[31] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. 2024. Schema Matching with Large Language Models: an Experimental Study. *arXiv preprint arXiv:2407.11852* (2024).

[32] Derek Paulsen, Yash Govind, and AnHai Doan. 2023. Sparkly: A simple yet surprisingly strong TF/IDF blocker for entity matching. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1507–1519.

[33] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[34] Fotis Psallidas and Eugene Wu. 2018. Smoke: Fine-grained lineage at interactive speed. *arXiv preprint arXiv:1801.07237* (2018).

[35] Egor Pushkin. 2020. Theoretical Model and Practical Considerations for Data Lineage Reconstruction. *arXiv preprint arXiv:2001.11506* (2020).

[36] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2020. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2010.08191* (2020).

[37] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal* 10 (2001), 334–350.

[38] Christopher Ré and Dan Suciu. 2008. Approximate lineage for probabilistic databases. *Proceedings of the VLDB Endowment* 1, 1 (2008), 797–808.

[39] Mohammed Suhail Rehman. 2023. *Reconstructing the Lineage of Artifacts in Data Lakes*. Ph.D. Dissertation. The University of Chicago.

[40] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[41] Alexander Schoenenwald, Simon Kern, Josef Viehhauser, and Johannes Schildgen. 2021. Collecting and visualizing data lineage of Spark jobs: Digesting Spark execution plans to surface lineage graphs via a full-stack application. *Datenbank-Spektrum* 21 (2021), 179–189.

[42] Pavel Shvaiko and Jérôme Euzenat. 2005. A Survey of Schema-Based Matching Approaches. *J. Data Semant.* (2005), 146–171.

[43] Guido De Simoni, Anurag Raj, Melody Chien, and Stephen Kennedy. 2025. *Magic Quadrant for Data and Analytics Governance Platforms*. Technical Report ID G00807073. Gartner.

[44] Edwin Soedarmadji, Helge S Stein, Santosh K Suram, Dan Guevarra, and John M Gregoire. 2019. Tracking materials science data lineage to manage millions of materials experiments and analyses. *npj Computational Materials* 5, 1 (2019), 79.

[45] Nandan Thakur, Nils Reimers, Johannes Daxenberger, and Iryna Gurevych. 2020. Augmented SBERT: Data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. *arXiv preprint arXiv:2010.08240* (2020).

[46] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. 2021. Deep learning for blocking in entity matching: a design space exploration. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2459–2472.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[48] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808* (2020).

[49] Jing Zhang, Bonggun Shin, Jinho D Choi, and Joyce C Ho. 2021. SMAT: An attention-based deep learning solution to the automation of schema matching. In *Advances in Databases and Information Systems: 25th European Conference, ADBIS 2021, Tartu, Estonia, August 24–26, 2021, Proceedings 25*. Springer, 260–274.

[50] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M Patel. 2023. Schema matching using pre-trained language models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1558–1571.

[51] Wayne Xin Zhao, Jing Liu, Ruiyang Ren, and Ji-Rong Wen. 2024. Dense text retrieval based on pretrained language models: A survey. *ACM Transactions on Information Systems* 42, 4 (2024), 1–60.